**Online Shopping Cart**

**A Project Report Submitted in Partial Fulfillment of the Requirements for the Degree**

**Of**

**Bachelor of Technology**

**In**

**Computer Science and Engineering**

**By**

### Amit Kumar (11162580)

### Deep Shikha ( 11162588)

### Raghvendra Tiwari (11162595 )

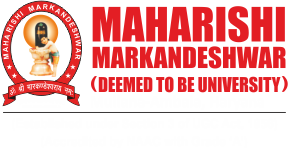
### Shreyasi (11166101 )

Under the supervision of

### Ms. Jasleen Kaur

**Assistant Professor**

**Department of Computer Science & Engineering**



**M.M.EngineeringCollege, Mullana, Ambala**

**Maharishi Markandeshwar (Deemed to be University), Mullana, Ambala, Haryana, India**

**December 2018**

**Online Shopping Cart**

**A Project Report Submitted in Partial Fulfillment of the Requirements for the Degree**

**Of**

**Bachelor of Technology**

**In**

**Computer Science and Engineering**

**By**

### Amit Kumar ( 11162580 )

### Deep Shikha ( 11162588 )

### Raghvendra Tiwari ( 11162595 )

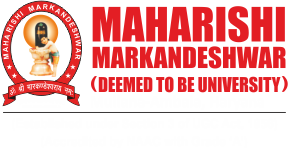
### Shreyasi ( 11166101 )

Under the supervision of

### Ms. Jasleen Kaur

**Assistant Professor**

**Department of Computer Science & Engineering**



**M. M.EngineeringCollege, Mullana, Ambala**

**Maharishi Markandeshwar (Deemed to be University), Mullana, Ambala, Haryana, India**

**December 2018**

**Department of Computer Science &Engineering**

**M.M.EngineeringCollege, Mullana, Ambala, Haryana, India**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Candidate Declaration**

### I hereby certify that the work which is being presented in the project entitled “Online Shopping Cart” in fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering of M.M Engineering College, Mullana, Ambala, Haryana, India is an authentic record of my own work carried out during a period from July 2018 to December 2018, under the supervision of Ms. Jasleen Kaur (Assistant Professor). The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other Institute/University.

**Amit kumar (11162580)**

**Deep Shikha (11162588)**

**Raghvendra Tiwari(11162595)**

**Shreyasi (11166101)**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: ***Ms. Jasleen Kaur***

**Assistant professor**

**Department of Computer Engineering**

**Head of Department**

**Acknowledgement**

I wish to express my deep sense of indebtedness and sincerest gratitude to my guide**, Ms. Jasleen Kaur (Assistant Professor), Department of Computer Science & Engineering, M. M. Engineering College, Mullana, Ambala, Haryana**, for his invaluable guidance and constructive criticism throughout this dissertation. He has displayed unique tolerance and understanding at every step of progress and encourages me. I deem it my privilege to have carried out my Dissertation work under his able guidance.

I would especially like to thank **Dr. Sandip Kumar Goel (Professor and Head), and Coordinator Dr Suneet Kumar (Associate Professor) Department of Computer Science & Engineering, M.M Engineering College, Mullana, Ambala,** without whom, this work would not have been as it is now.

As a Final Personal Note, I am grateful to my parents, who are inspirational to me in their understanding, patience and constant encouragement.

**Amit kumar (11162580)**

**Deep Shikha (11162588)**

**Raghvendra Tiwari (11162595)**

**Shreyasi (11166101)**

**Abstract**

The business-to- consumer aspect of an online shopping is the most visible business use of the World Wide Web. The primary goal of an online shopping site is to sell goods and services online. This project deals with developing an e-commerce website for online shopping. It provides the user with a catalogue of different goods and services available for purchase in the store. In order to facilitate online purchase a shopping cart is provided to the user. The system is implemented using a 3- tier approach, with a backend database, a middle tier of Microsoft Internet Information Services (MIIS) and PHP, and a web browser as the front end client. In order to develop an e- commerce website for online shopping, a number of Technologies must be studied and understood. These include multi- tiered architecture, server and client side scripting techniques, implementation technologies such as ASP.NET, programming language (such as C,C#, JAVA), relational databases (such as MySQL,H2, Access). This is a project with the objective to develop a basic website where a consumer is provided with a shopping cart application and also to know about the technologies used to develop such an application. This document will discuss each of the underlying technologies to create and implement an e-commerce website for online shopping.

**LIST OF TABLES**

TABLE 1 User detail table………………………………………………….24

TABLE 2 User detail description table……………………………………..25

TABLE 3 User authentication table………………………………………...26

**TABLE 4** User authentication table description…………………………....26

TABLE 5 Login module table……………………………………………...40

TABLE 6 Register, login and logout module integrated table……..………41

**LIST OF FIGURES**

FIGURE 1.1 Request-Response Cycle for JSP Page…………………...………...3

FIGURE 1.2 Java Application……………………………………………………4

FIGURE 1.3 JDBC Architecture…………………………………………………5

FIGURE 2.1 Software Development Life Cycle………………………………..13

FIGURE 3.1 Steps Followed for requirement…………………………………..16

FIGURE 5.1 Functional Decomposition Diagram……………………………...27

FIGURE 6.1 Customer Browser Context DFD…………………………………29

FIGURE 6.2 Customer ShoppingCart Context DFD …………………………..29

FIGURE 6.3 Customer Shopping Cart Detailed DFD……………..……………30

FIGURE 6.4 Customer Authentication Context DFD…………………………..30

FIGURE 6.5 Customer Authentication User Profile DFD………………………30

FIGURE 6.6 Authentication User Purchase Context DFD………………………31

FIGURE 6.7 Authentication User Purchase DFD……………………………….32

FIGURE 7.1 Login Page………………………………………………………...33

FIGURE 7.2 Sign Up Page………………………………………………………34

FIGURE 7.3 Confirmation of Sign Up Detail……………………………………35

FIGURE 7.4 Welcome Page……………………………………………………..36

FIGURE 7.5 Home Page…………………………………………………………36

FIGURE 7.6 Product Detail……………………………………………………...37

FIGURE 7.7 Product in Cart…..…………………………………………………

TABLE OF CONTENTS

Title……………………………………………………………………………...ii

Candidate’s Declaration………………………………………………………...iii

Acknowledgement………………………………………………………………iv

Supervisor Performa…………………………………………………………….v

Abstract…………………………….………………………………………………………….vi

List of Tables…………………………………………………………………...vii

List of Figures……………………………………………………………….....viii

[INTRODUCTION……………………………………………………………….1](file:///E:\Project\Project%20report.docx#_Toc278093123)

1.1 Project Description……………………………………………………………………………………………………………….2

1.2 Technology Used………………………………………………………………………………………………………………….2

1.2.1 Java Server Pages……………………………………………………………………………………….……2

1.2.2 JDBC………………………………………………………………………………………………………….…….4

1.2.3 JavaScript…………………………………………………………………………………………………….….7

1.2.4 H2…………………………………………………………………………………………………………………..8

1.2.5 BootStrap………………………………………………………………………………………………………..9

1.2.6 Angular JS………………………….…………………………………………………………………………..11

PROJECT LIFE CYCLE……………………………………………………….12

2.1 Software Development Life Cycle……………………………………………………………………………………12

REQIREMENT ANALYSIS……………………………………………………16

3.1 Overview………………………………………………………………………………………………………………………………16

3.2Objective of requirement analysis…………………………………………………………………………………………17

3.3 Software Requirement Specification…………………………………………………………………………………….17

3.3.1 Introduction……………………...…………………………………………………………………………..…17

3.3.2 Production Description……………………………………………………………………………….18

3.3.3 Usage Scenario…………………………………………….……………………………………………..19

3.3.4 Data Object and description……………………………………………………………………….20

REQUIREMENT (SOFTWARE/ HARDWARE)……………………….......21

SYSTEM DESIGING………………………………………...……………...22

5.1 Overview……………………………………..………………...……………...22

5.1.1Database Design……………………………………….…………….22

5.1.2 Human Machine Interface Design…………………….…………….22

5.2 Software Design Specification………………………………………………..23

5.3 Process Model………………………………………………………………...26

5.3.1 Functional Decomposition Diagram………………………………..26

DATA FLOW DIAGRAM (DFD)………………………………………….28

APPLICATION SCREENSHOTS………………………………………….33

TESTING AND IMPLEMENTATION…………………………………….39

8.1Testing…………………………………………………………….39

8.1.1 Testing Objectives………………………………………39

8.1.2 Testing Principle………………………………………...39

8.1.3 Unit Testing……………………………………………..40

8.1.4 Integration Testing………………………………………40

8.1.5 System Testing…………………………………………..41

8.1.6 Optimization Points……………………………………...41

FUTURE SCOPE……………………………………………………………43

REFERENCE………………………………………………………………..45

# APPENDIX A: SOURCE CODE…………………………………………………………47

# 

# CHAPTER – 1

**ONLINE SHOPPING CART**

**1. INTRODUCTION**

At the very core of an online storefront is the shopping cart system that enables customers to find and purchase products and services.

The system allows customers to acquire a list of items for purchase by placing items into a virtual cart. At the checkout the software will calculate a total of the invoice with any applicable taxes, including shipping and handling.

Currently there are three different types of shopping cart systems that are built to better suit the needs of ecommerce stores; ordering systems, online storefront builder systems and specialized systems. An ordering shopping system is the most basic that makes it possible to select and purchase online. For larger store offerings, a Storefront builder shopping cart system combines all of the features of an order system combined with contact management system (CMS). This allows you a greater level of control over your store content. However, if your ecommerce store offers business to business (b2b) solutions then a specialized ecommerce shopping system would likely be the best fit.

Such shopping systems normally include a database, a storefront and an administrative area. The database stores customer data, order information, product details, etc. A storefront displays this information while an administration area allows a business to manage the ecommerce store. Storefronts should include a store catalog that is presented usefully and is pleasing for customers. Good administrative functionality should be easy to navigate, manage modules, assign options, and flexible to manage multiple shipping needs.

Many online ecommerce store builders will offer either a licensed or hosted shopping cart system. A licensed shopping cart system can be downloaded and installed on a web server often associate with a onetime fee. Since the merchant will own the license the system can be hosted on any web server. A hosted system can never be downloaded but is hosted instead through a service provider and requires a recurring fee or nominal percentage of sales.

Merging the shopping system with your ecommerce store can be easy. Shopping systems are typically able to create dynamic pages. These pages are generated using information in real time from the database that contains product information and store settings. For example a "Discount" page could contain link to a page created by the shopping cart that shows discounted products for that day or month. Some system use pre-formatted templates to display ecommerce information. Other shopping cart systems do not use templates and allow for a custom more unique graphical interface design.

A good shopping cart system enables both the ecommerce store and customer to participate in business with each other. Without it, ecommerce would not be possible and once you have set up the right shopping cart system, then you can [make money online](http://www.makemoneyonlinez.com/) without too much fuss.

* 1. **Project Description**

E-commerce is fast gaining ground as an accepted and used business paradigm. More and more business houses are implementing web sites providing functionality for performing commercial transactions over the web. It is reasonable to say that online shopping system project report the process of shopping on the web is becoming commonplace.

The objective of this online shopping system project project is to develop a general purpose e-commerce store where any product (such as books, CDs, computers, mobile phones, electronic items, and home appliances) can be bought from the comfort of home through the Internet.

However, for implementation purposes, this online shopping system project paper will deal with an online book store.

An online store is a virtual store on the Internet where customers can browse the catalog and select products of interest. The selected items may be collected in a shopping cart. At checkout time, the items in the shopping cart will be presented as an order. At that online shopping system project report time, more information will be needed to complete the transaction. Usually, the customer will be asked to fill or select a billing address, a shipping address, a shipping option, and payment information such as credit card number. An e- mail notification is sent to the customer as soon as the order is placed.

* 1. **Technologies Used**
     1. **Java Server Pages (JSP)**

With the advent of Internet, the monolithic application architecture changed to the multi-tiered client/server architecture. The need for server-side scripting gradually began to dominate aspects of Web Programming. Microsoft introduced Active Server Pages (ASP) to capture the market demand for server-side scripting. Working on similar lines, Sun Microsystems released Java Server Pages (JSP) to add server side programming functionalities to Java.

A typical web application consists of the presentation logic representing the static content used to design the structure of a web page in terms of the page layout, color, and text. The business logic or the dynamic content involves application of business intelligence and diagnostics in terms of financial and business calculations. When developing web applications, time is often lost in situations where the developer is required to code for the static content.

JSP Technology has facilitated the segregation of work profiles of a web designer and a web developer. A Web Designer can design and formulate the layout for the web page by using HTML. On the other hand, a Web Developer working independently can use Java code and other JSP specific tags to code for the business logic. The simultaneous construction of static and dynamic content facilitates development of quality applications with increased productivity.

**The JSP Request-Response Cycle*:***JSP files are stored on the web server with an extension of .jsp. When the client/browser requests for a particular JSP page, the server in turn sends a request to the JSP Engine. The following figure represents the process of the flow of events that online shopping system project report occur after a client requests for a JSP page.

Figure 1.1 Request-Response Cycle for a JSP Page

The request-response cycle essentially comprises of two phases, namely the translation phase and request-processing phase. The translation phase is implemented by the JSP engine and involves generation of a servlet. Internally, these results in the creation of a class file for the JSP page that online shopping system project report implements the servlet interface. During the request-processing phase, the response is generated according to the request specification. The servlet then sends back a response corresponding to the request received. After the servlet is loaded for the first time, it remains active and possesses all the subsequent requests with responses, saving time that online shopping system project report would otherwise be lost in reloading a servlet at each request.

* + 1. **JDBC**

JDBC (Java Database Connectivity) is an application program interface (API) specification for connecting programs written in Java to the data in popular database. It is provided by Sun Microsystems, the application program interface lets you encode access request statements in structured query language (SQL) that online shopping system project report are then passed to the program that online shopping system project report manages the database. It returns the results through a similar interface.

From the user’s point of view, Java application looks something like this online shopping system project

Figure 1.2 Java Application

**Features of JDBC API:**

Java Database Connectivity (JDBC) provides a database programming API for Java programs. Some of the features of JDBC API are as follows:

* Contains a set of classes and interfaces that online shopping system project report are used to connect to a database built using any DBMS/RDBMS, submit SQL queries to a database, and retrieve and process the results of SQL queries.
* Is a low-level interface in which SQL *select* and *update* statements are called directly from within Java programs.
* Can be used with both two-tier and three-tier database architectures. In two-tier architecture, a Java program invokes the methods of JDBC API, which in turn communicates with the database server. In three-tier architecture, a Java applet or an HTML form submits SQL queries to a middle-tier server. Middle-tier server in turn uses JDBC API to communicate with the database server.

**JDBC Architecture:**

The JDBC architecture is based on a collection of Java interfaces and classes that online shopping system project report together enables you to connect to data sourced, to create and execute SQL statements, and to retrieve and modify sata in a database. These operations are illustrated in the figure below:



Figure 1.3 JDBC Architecture

Each of the boxes in the illustration represents a JDBC class or interface that online shopping system project report has fundamental role in accessing a relational database.

**JDBC Drivers:**

JDBC API takes care of converting Java commands to generic SQL statements. However, to address specific database issues, each database vendor provides a driver along with the database. Java Applications invoke the methods JDBC API. JDBC API in turn uses a driver to communicate with a specific database.

JDBC API submits queries to the JDBC driver. The JDBC driver converts queries to a form that online shopping system project report a particular DBMS/RDBMS can understand. The JDBC driver also retrieves the results of SQL queries, converts it into equivalent JDBC API classes and objects that online shopping system project report can be used by the application. Since the JDBC Driver only takes care of the interactions with database, any change made to the database does not affect the application.

**Java Application** **JDBC**

**API**

**JDBC DBMS/RDBMS**

**Driver**

**JDBC Architecture**

There are several categories of JDBC Drivers provided by different database vendors. They are:

* **JDBC-ODBC Bridge Driver:** The first category of JDBC drivers provides a bridge between the JDBC API and the ODBC API. There are several DBMS/RDBMS, such as MS Access and SQL Server that online shopping system project report contain the ODBC Driver embedded into them. Since the ODBC API is written in the C language and makes use of pointers and other constructs that online shopping system project report Java does not support, a Java program cannot directly communicate with an ODBC Driver. The bridge translates the standard JDBC calls to corresponding ODBC calls, and sends them to ODBC data source via ODBC libraries.
* **Native API Partly Java Driver:** These drivers use a mixture of Java implementation and vendor specific native APIs to provide data access. JDBC database calls are translated into vendor specific API calls. The database will process the request and sends the result back through the API, which will in turn forward them back to the JDBC driver. The JDBC driver will translate the result to the JDBC standard and return them to the Java application. There is one layer fewer to go through than for a type 1 driver and so in general a type 2 driver will be faster than a type 1 driver.

Some DBMS/RDBMS such as DB2 and Informix contain a JDBC driver supplied by the database vendor.

* **Intermediate Database Access Server:** Type 3 drivers use an intermediate database server that online shopping system project report has the ability to connect multiple Java clients to multiple database servers. Clients connect to database server via an intermediate server component that online shopping system project report acts as a gateway for multiple database servers. The java client application sends a JDBC call through a JDBC driver to the intermediate data access server, which completes the request to the data sourcing using another driver ( for example, a type 2 driver).

BEA WebLogic includes a type 3 driver. One of the benefit of using a type 3 driver is that online shopping system project report it allows flexibility on the architecture of the application, as the intermediate server can can abstract details of connection to database servers.

* **Native Protocol Pure Java Driver/JDBC-Net Pure Java Driver:** These drivers convert the JDBC API calls to direct network calls using vendor specific networking protocol. They do this online shopping system project by making direct socket connections with the database. Type 4 drivers offer better performance than others.

**1.2.3.JavaScript**

The project uses JavaScript as the client side scripting language for JSP/HTML pages in the project.

JavaScript is an easy to use object-scripting language designed for creating live online applications that online shopping system project report link together resources on both clients and servers. JavaScript is designed for use by HTML page authors and enterprise application developers to dynamically script the behavior of objects running on either the client or the servers. JavaScript ‘s design and concept represent the next generation of software for the Internet and is:

* Designed for creating network centric applications
* Complementary to and integrated with Java
* Complementary to and integrated with HTML
* Open and cross platform

JavaScript is a platform-independent, event driven, interpreted programming language developed by Netscape Communications Corp. and Sun Microsystems. Originally called Livescript, JavaScript is a programming language that online shopping system project report can be included on web pages to make them more interactive.

JavaScript is easier to understand, less complex version of its distant cousin, Java. It is a text-based language that online shopping system project report must be placed within HTML that online shopping system project report must be placed within, HTML, to be read by the browser and interpreted so the instructions can be performed.

JavaScript is a preferred language for client side scripting. This online shopping system project is mainly because when client side scripting is done browser compatibility is an issue of concern and both major browsers support JavaScript.

**1.2.4 H2 :**

H2 is an open-source lightweight Java database. It can be embedded in Java applications or run in the client-server mode. Mainly, H2 database can be configured to run as inmemory database, which means that data will not persist on the disk. Because of embedded database it is not used for production development, but mostly used for development and testing.

This database can be used in embedded mode or in server mode. Following are the main features of H2 database −

* Extremely fast, open source, JDBC API
* Available in embedded and server modes; in-memory databases
* Browser-based Console application
* Small footprint − Around 1.5MB jar file size

## Features of H2 Database

The main features of H2 Database are as follows −

* It is an extremely fast database engine.
* H2 is open source and written in Java.
* It supports standard SQL and JDBC API. It can use PostgreSQL ODBC driver too.
* It has embedded and Server mode.
* H2 supports clustering and multi-version concurrency.
* It has strong security features.

## Additional Features

Following are some additional features of H2 Database −

* H2 is a disk-based or in-memory databases and tables, read-only database support, temporary tables.
* H2 provides transaction support (read committed), 2-phase-commit multiple connections, table level locking.
* H2 is a cost-based optimizer, using a genetic algorithm for complex queries, zeroadministration.
* H2 contains scrollable and updatable result set support, large result set, external result sorting, functions can return a result set.
* H2 supports encrypted database (AES), SHA-256 password encryption, encryption functions, and SSL.

## Components in H2 Database

In order to use H2 Database, you need to have the following components −

* A web browser
* A H2 console server

This is a client/server application, so both server and client (a browser) are required to run it.

**1.2.5BOOTSTRAP:**

Bootstrap is a sleek, intuitive, and powerful, mobile first front-end framework for faster and easier web development. It uses HTML, CSS and Javascript.

## History

Bootstrap was developed by **Mark Otto** and **Jacob Thornton**at Twitter. It was released as an open source product in August 2011 on GitHub.

## Uses Bootstrap

* **Mobile first approach** − Bootstrap 3, framework consists of Mobile first styles throughout the entire library instead them of in separate files.
* **Browser Support** − It is supported by all popular browsers.
* **Easy to get started** − With just the knowledge of HTML and CSS anyone can get started with Bootstrap. Also the Bootstrap official site has a good documentation.
* **Responsive design** − Bootstrap's responsive CSS adjusts to Desktops, Tablets and Mobiles. More about the responsive design is in the chapter [Bootstrap Responsive Design.](https://www.tutorialspoint.com/bootstrap/bootstrap_responsive_utilities.htm)
* Provides a clean and uniform solution for building an interface for developers.
* It contains beautiful and functional built-in components which are easy to customize.
* It also provides web based customization.
* And best of all it is an open source.

## Bootstrap Package

* **Scaffolding** − Bootstrap provides a basic structure with Grid System, link styles, and background. This is is covered in detail in the section Bootstrap Basic Structure
* **CSS** − Bootstrap comes with the feature of global CSS settings, fundamental HTML elements styled and enhanced with extensible classes, and an advanced grid system. This is covered in detail in the section Bootstrap with CSS.
* **Components** − Bootstrap contains over a dozen reusable components built to provide iconography, dropdowns, navigation, alerts, pop-overs, and much more. This is covered in detail in the section Layout Components.
* **JavaScript Plugins** − Bootstrap contains over a dozen custom jQuery plugins. You can easily include them all, or one by one. This is covered in details in the section Bootstrap Plugins.
* **Customize** − You can customize Bootstrap's components, LESS variables, and jQuery plugins to get your very own version.

**1.2.6 Angular JS**

AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. AngularJS's data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner with any server technology.

AngularJS is what HTML would have been, had it been designed for applications. HTML is a great declarative language for static documents. It does not contain much in the way of creating applications, and as a result building web applications is an exercise in what do I have to do to trick the browser into doing what I want?

The impedance mismatch between dynamic applications and static documents is often solved with:

* **a library** - a collection of functions which are useful when writing web apps. Your code is in charge and it calls into the library when it sees fit. E.g., jQuery.
* **frameworks** - a particular implementation of a web application, where your code fills in the details. The framework is in charge and it calls into your code when it needs something app specific. E.g., durandal, ember, etc.

AngularJS takes another approach. It attempts to minimize the impedance mismatch between document centric HTML and what an application needs by creating new HTML constructs. AngularJS teaches the browser new syntax through a construct we call *directives*. Examples include:

* Data binding, as in {{}}.
* DOM control structures for repeating, showing and hiding DOM fragments.
* Support for forms and form validation.
* Attaching new behavior to DOM elements, such as DOM event handling.
* Grouping of HTML into reusable components.

# CHAPTER – 2

**PROJECT LIFE CYCLE**

**2.1Software Development Life Cycle:**

Software development organization follows some process when developing a software product. A key component of any software development process is the life cycle model on which the process is based. The particular life cycle model can significantly affect overall life cycle costs associated with a software product. Life cycle of the software starts from concept exploration and at the retirement of the software.



Figure 2.1 Software Development Life Cycle

**PHASES OF SYSTEM DEVELOPMENT LIFE CYCLE:**

The system development life cycle is classically thought of as the set of activities that online shopping system project report analysts, designers and users carry out to develop and implement an information system. The system development life cycle consists of the following activities:

* Preliminary investigation.
* Requirement Analysis.
* System Designing.
* Coding.
* System Testing.
* Implementation and Maintenance.
* **Preliminary Investigation: -**

An important outcome of the preliminary investigation is the determination that online shopping system project report system is feasible or not. In the conduct of feasibility study, there are three major distinct and interrelated areas were taken into consideration. They are as follows:

1. Technical Feasibility :

The System of operation which was functioning earlier was totally manual, with no kind of automation or computerization. All the departments were maintaining separate registers for keeping various records. Due to expansion of schools more workspace and it appears a tedious task to maintain with specifying equipment and software that online shopping system project report will successfully support the tasks required. As a result the computerized system is technically feasible as it is efficient, less time consuming, can produce outputs faster, can input large amount of data in limited time scale and easier to use in operation

1. Operational Feasibility : -

The ultimate users i.e. the people who are supposed to use the system are trained for a period of one month so as to get familiar with the new system and its operation. They are taught about the new skills and the new technology and how the technology will be useful to them in their functioning. Operational feasibility is concerned with human, organizational and political aspects. General impression of these factors is gained from the corporate appraisal.

1. Economical Feasibility :

The computerized system is economically feasible in the sense the cost of the hardware and software and the cost to training of personnel of the company to operate the system and the installation cost is less than the cost of maintaining the registers. This online shopping system project may not be a big sum in the long run of the school business. Also the time taken for the entire process of formulation, checking, studying and installation of the project has been equal to one working month of the school. As a result, there has been no hesitation on pert of the management in adopting the new system.

* **Requirements analysis:-**

Analysis of requirements includes studying the existing system and collecting data. During analysis, data are collected on the available files, decision points and transaction handled by the present system. Once the structured analysis is completed, the analyst has affirmed understanding of what is to be done.

* **System Designing:**

The design of an information system produces the details that online shopping system project report clearly describe how a system will meet the requirements identified during system analysis. System analysts begin the design process by identifying reports and other outputs system will produce. The system design also describes the data to be input, calculated or stored.

* **Coding: -**

This online shopping system project is the phase in which computer based system is constructed from the specifications prepared in the design phase. Equipment is acquired and installed during the development phase. All necessary procedure, manuals software specifications, and other documentation are completed. The staff is trained.

* **System Testing:-**

During system testing, the system is used experimentally to ensure that online shopping system project report the software does not fail. In other words we can say that online shopping system project report it will run according to its specifications and in the way users expect. Special test data are input for processing, and the result examined.

* **Implementation, Evaluation and Maintenance:-**

Implementation is the process of having systems personnel check out and put new equipments into use, train users, install the new application and construct any files of data needed to use it.

Evaluation of the system is performed to identify its strength and weaknesses. Maintenance is necessary to eliminate errors in the working system during its working life and to tune the system to any variations in its working environment. The importance of maintenance is to continue to bring the new system to standards.

# 

# CHAPTER – 3

**REQUIREMENT ANALYSIS**

**3.1 Overview:**

Analysis is a Fact Finding Technique where studies like User’s need, System Requirement Specifications, Feasibility Analysis and Cost-Benefit Analysis are carried out.

This online shopping system project is the most important step in a software project where we get

Figure 3.1 Steps followed for requirement

a general idea about the needs of the customers or end users by having man to man conversation

with them and about the various conditions and restrictions that online shopping system project report have to be taken care of while developing the software application.

The purpose of this online shopping system project phase is to identify, analyze and document the exact requirements for the system. The developer, customer, a marketing organization, or any combination of the three may perform such study. It is extremely important that online shopping system project report the developers of the system study the existing system thoroughly otherwise it is impossible to satisfy the needs of the user. The requirements at this online shopping system project stage are in end-user terms.

During the Requirement Analysis Phase, the development team analyzes the requirements to be fulfilled by the Online Shoping Cart website and identifies the probable approach for meeting these requirements. To identify the requirements needed by the website, we decided to study the existing Shopping Cart process like Searching Book, Adding to Cart, money transfer. In this online shopping system project phase we have also collect necessary information regarding the details to be stored Registered customer.

# 3.2 Objective of Requirement Analysis:

# Requirement analysis was conducted with the following objectives in mind:

Identification of need

Information Gathering

Evaluate the system concept of feasibility

### 3.2.1*.* Identification of Need:

### The success of the system depends largely on how accurately a problem is defined, thoroughly investigated, and properly carried out through the choice of solution. Users need identification and analysis is concerned with what the user needs rather then what he/she wants. Until the problem has been identified, defined, and evaluated the analyst shouldn't think about solutions and whether the problem is worth solving or not.

#### 3.2.2. Information Gathering:

A key part of system development is gathering information. The analyst must know what information to get, where to find it, how to collect it, and how to make use of it. The proper use of tools for gathering information is the key to successful analysis.

3.2.3. Feasibility Study:

Feasibility study is carried out to test if the proposed system is feasible in terms of economy, technology, resource availability etc. As such, given unlimited resources and infinite time, all projects are feasible. Unfortunately, such results and time are not possible in real life situations. Hence it is both necessary and prudent to evaluate the feasibility of the project at the earliest possible time in order to avoid unnecessary wastage of time, effort and professional embarrassment over an ill conceived system.

**3.3 Software Requirements Specification (SRS):**

**3.3.1 Introduction:**

1. **Purpose:**

This online shopping system project document completely describes *what* the “Shopping Cart” should do without describing *how* the software will do it. The basic goal of the requirement phase is to produce the SRS, which describing the complete external behavior of the purposed software.

1. **Scope:**

This online shopping system project document is the only one that online shopping system project report describes the requirements of the system. It is meant for use by the developer and will be the basis for validating the final delivered system. Any changes made to the requirements in the future will have to go through a formal changes approval process. The developer is responsible for asking for clarifications, where necessary, and will not make any alteration without the permission of the client.

1. **Developer’s responsibility:**

The developer is responsible for:

(a) Developing the system.

(b) Installing the software on the client’s hardware.

(c) Conducting any user training that online shopping system project report might be needed for using the system.

(d) Maintaining the system for a period of one year after installation.

# 3.3.2 Product description:

This online shopping system project section provides an overview of the software. This online shopping system project section describes the goal and objective of the software. This online shopping system project section also briefly describes the general requirements of the software. This online shopping system project section is very important for the verification of the software after the completion whether the objective and requirements of the software will met or not.

# Goals and objective:

The main purpose of “Online shopping Cart” is to provide the Shopping related services on the Internet. This online shopping system project software also helps to automate the process of ordering the books in home using internet . The goals of “ Online Shopping Cart” are:

* To automate the time consuming process to go to book store and purchases books .
* To advertise the new books available in Internet.
* To manage the records of customers, Books Details, Stock Details.
* To provide a searchable database of all customers and accounts.
* To minimize the amount of paper work required in the daily services.
* To provide a secure interface for the banking transactions.
* To provide an interface so that online shopping system project report user can take advantage of anytime, anywhere Shopping.

1. **General requirements:**

During the Requirement Analysis Phase, the development team analyzes the requirements to be fulfilled by the Online Shopping website and identifies the probable approach for meeting these requirements. To identify the requirements needed by the website, we decided to study the existing Shoping process like Searching Books , money transfer. In this online shopping system project phase we have also collect necessary information regarding the details to be stored by the database for opening an account.

Finally, it was identified that online shopping system project report the Shopping Website should:

* Enable the visitors to fill Registration form.
* Provide details of the various Books available in Stores.
* Provide the information about the rate of the available books.
* Be secure enough against the malicious security attack, identity verification of the registered user and authorization.
* Be able to handle various run time exceptions and errors.
* It should provide proper interfaces to manage and view details.
* The web pages should be user friendly and well design to attract visitors.

**3.3.3 Usage Scenario:**

This online shopping system project section provides a usage scenario for the software. It organized information collected during requirements elicitation into use-cases.

## User Profiles:

There will be three levels of users:

* Administrator level (Employee)
* User level (Account Holder)

## Use-cases:

### Administrator level:

This online shopping system project level of users will be able to insert new Books, new Category and Price information,. They will also be able to generate

**User Level:**

This online shopping system project level of users will be able to Search the books, Order the books etc. They can view their records.

**3.3.4 Data Objects and description:**

During the requirement analysis phase, the development team examines existing Shopping Cart . After examining all process and feasibility we decided to consider following points while designing database:

* It should store information of the Registered user details.
* It should store information of the Books details.
* It should store information of the various accounts.
* It should store information of the various category of the books.

# CHAPTER – 4

**REQUIREMENTS**

##### Hardware Requirement:

##### 

**I. Server (Windows 2000 Server (NT))**

Microsoft 2000 Server is based on NT Technology biased to run server side scripts in ASP technology.

Disk Space 3 GB.

Web Server Microsoft IIS (6.0)

#### II. Client

Disk Space 1GB.

Processor Pentium III

Processor Speed 1.13Ghz

Memory 256 MB

#### Software Requirement:

Operating System : Windows / NT / XP

Technologies : JDBC, Java Beans

Client Side Scripting Language : HTML and JSP and Java.

Server Side Scripting Language : JSP

Markup Language : HTML

Database Server : H2

Web Server : Tomcat 4.0

# CHAPTER – 5

**SYSTEM DESIGING**

* 1. **OVERVIEW:**

System design is a solution, a “HOW TO APPROACH” to the creation of a new system. This online shopping system project important phase is composed of several steps. It provides the understanding and procedural details to implement the system. Design goes through a logical and physical stage of the progress. Logical design reviews the present physical system, prepares input/output specifications, makes audit security and control specifications, detailed implementation plans, and prepares the logical design walkthrough. The physical design makes out the details of the physical system, plans the system implementation and specifies any new hardware and software products.

**5.1.1 DATABASE DESIGN:**

The collection of data is usually referred to as the database. The database contains the information about one particular enterprise. Database system of data involves both the definitions of structures for the storage of information, processing and mechanism for the manipulation of information. In addition, the database system provides for the safety of information stored in the database despite system crashes or attempts of unauthorized access.

## HUMAN-MACHINE INTERFACE DESIGN:

The design of the human machine interface in one of the most important aspects of system design. A good interface design should take into account the following factors: -

**User characteristics:**

It includes consideration of the kinds of the users who will use the equipment, their diverse backgrounds and skills, the user expectations as well as their physical characteristics. The users who posses high degree of skill often prefer more powerful functions which usually means greater complexity, unskilled operations, on the other hand, would simple functions which are easier to learn and use.

**Task Characteristics:**

The nature of the users tasks differ and therefore the needs for specific-kinds of service from the system. For example unstructured tasks usually requires a more flexible mode of interaction to meet the varying needs of users as opposed to structured tasks, which are more predictable and repetitive. The sequence and frequency with which certain tasks are performed will also affect the optimal design of the user interface.

**Functional Characteristics:**

It refers to the various functions required to perform the tasks and the ease with which these functions can be learnt made use of by the users while ascertaining the functional characteristics, the support facilities required to perform the functions, also need to be taken into account. These include facilities like training, on line help, documentation, expert system etc. the other aspect of functional characteristics of a system its performance criteria like response time, fault tolerance etc.

**INPUT DESIGN:**

The most common cause of errors in data processing is inaccurate input data. Errors entered by data entry operators can be controlled by the input design. Input design is the process of converting user-oriented inputs to computer based formats. The goal of input design to make data entry easy logical and free from errors.

**OUTPUT DESIGN:**

Computers are the most important source of information to the user. Inputs are fed into computers to acquire the required outputs. The computers can provide valuable information’s in the form of well-documented outputs for various values. The major form of output is a hardcopy (reports) from the printer. Reports are around the output requirements of the user.

# Software Design Specification:

This online shopping system project section provides an overview of the entire design document. This online shopping system project document describes all data, architectural, interface and component-level design for the software.

## Table Definition:

## Table Name : USER\_Profile

Purpose : Store the information about all the registered user.

## Primary Key : UserName

Foreign Key :

**Columns Definition:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S. NO.** | **Name** | **Datatype** | **Size** | **Nulls?** | **Default Value** |
| 1 | **USERNAME** | VARCHAR2 | 30 | No |  |
| 2 | **PASSWORD** | VARCHAR2 | 15 | No |  |
| 3 | **FIRSTNAME** | VARCHAR2 | 10 | No |  |
| 4 | **MIDDLENAME** | VARCHAR2 | 10 | Yes |  |
| 5 | **LASTNAME** | VARCHAR2 | 10 | No |  |
| 6 | **ADDRESS1** | VARCHAR2 | 40 | No |  |
| 7 | **ADDRESS2** | VARCHAR2 | 40 | Yes |  |
| 8 | **CITY** | VARCHAR2 | 40 | No |  |
| 9 | **STATE** | VARCHAR2 | 20 | No |  |
| 10 | **PINCODE** | VARCHAR2 | 10 | No |  |
| 11 | **EMAIL** | VARCHAR2 | 25 | No |  |
| 12 | **PHONE** | VARCHAR2 | 12 | No |  |

Table 1: User detail table

**Table Description:**

|  |  |  |
| --- | --- | --- |
| **S. NO.** | **Name** | **description** |
| 1 | **USERNAME** | Store the user name |
| 2 | **PASSWORD** | Store information of user password |
| 3 | **FIRSTNAME** | Store the first name of the user |
| 4 | **MIDDLENAME** | Store the middle name of the user. |
| 5 | **LASTNAME** | Store the information of the user last name. |
| 6 | **ADDRESS1** | Store the information of the user address. |
| 7 | **ADDRESS2** | Optional |
| 8 | **CITY** | User city |
| 9 | **STATE** | User state |
| 10 | **PINCODE** | State pin code |
| 11 | **EMAIL** | User email address |
| 12 | **PHONE** | User phone number |

Table 2: User detail description table

**Table Definition:**

## Table Name : USER\_AUTH

Purpose : Stores the username and password of various end users.

Primary Key :

Foreign Key : USER\_PROFILE.USERNAME→USER\_AUTH.USERNAME

**Columns definition:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S. NO.** | **Name** | **Datatype** | **Size** | **Nulls?** | **Default Value** |
| 1 | **USERNAME** | VARCHAR2 | 30 | No |  |
| 2 | **PASSWORD** | VARCHAR2 | 15 | No |  |

Table 3: User Authentication table

**Table description:**

|  |  |  |
| --- | --- | --- |
| **S. NO.** | **Name** | **description** |
| 1 | **USERNAME** | Store the user name of the end users. |
| 2 | **PASSWORD** | Stores the password of the user. |

Table 4: User Authentication description table

**5.3. Process Model**

A Process Model tells us about how the data is processed and how the data flows

from one table to another to gather the required information. This online shopping system project model consists of the

Functional Decomposition Diagram and Data Flow Diagram.

**5.3.1. Functional Decomposition Diagram**

A decomposition diagram shows a top-down functional decomposition of a system and exposes the system's structure. The objective of the Functional Decomposition is to break down a system step by step, beginning with the main function of a system and continuing with the interim levels down to the level of elementary functions. The diagram is the starting point for more detailed process diagrams, such as data flow diagrams (DFD). Figure shows the Functional Decomposition Diagram for this online shopping system project project.

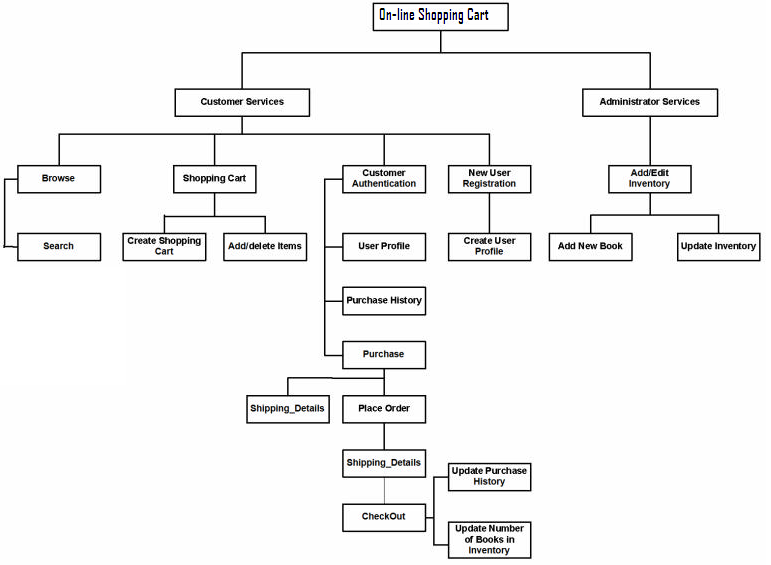


Figure 5.1: Functional Decomposition Diagram

# CHAPTER – 6

**DATA FLOW DIAGRAM**

**6.1 Data Flow Diagram (DFD)**

Data Flow Diagrams show the flow of data from external entities into the system,

and from one process to another within the system. There are four symbols for drawing a

DFD:

1. Rectangles representing *external entities*, which are sources or destinations of

data.

2. Ellipses representing *processes*, which take data as input, validate and process it

and output it.

3. . Arrows representing the *data flows*, which can either, be electronic data or

physical items.

4. Open-ended rectangles or a Disk symbol representing *data stores*, including electronic stores such as databases or XML files and physical stores such as filing cabinets or stacks of paper.

Data Flow Diagrams for the current system. Each process within the system is first shown as a Context Level DFD and later as a Detailed DFD. The Context Level DFD provides a conceptual view of the process and its surrounding input,output and data stores. The Detailed DFD provides a more detailed and comprehensive view of the interaction among the sub-processes within the system.



Figure 6.1: Customer-Browse Context DFD



Figure 6.2: Customer - ShoppingCart Context DFD



Figure 6.3: Customer - Shopping Cart Detailed DFD

******

Figure 6.4: Customer-Authentication Context DFD



Figure 6.5: Customer-Authentication-UserProfile DFD



Figure 6.6: Authenticated User-Purchase Context DFD



Figure 6.7: Authenticated User-Purchase DFD

# CHAPTER – 7

**APPLICATION SCREENSHOT**

**Login Page**

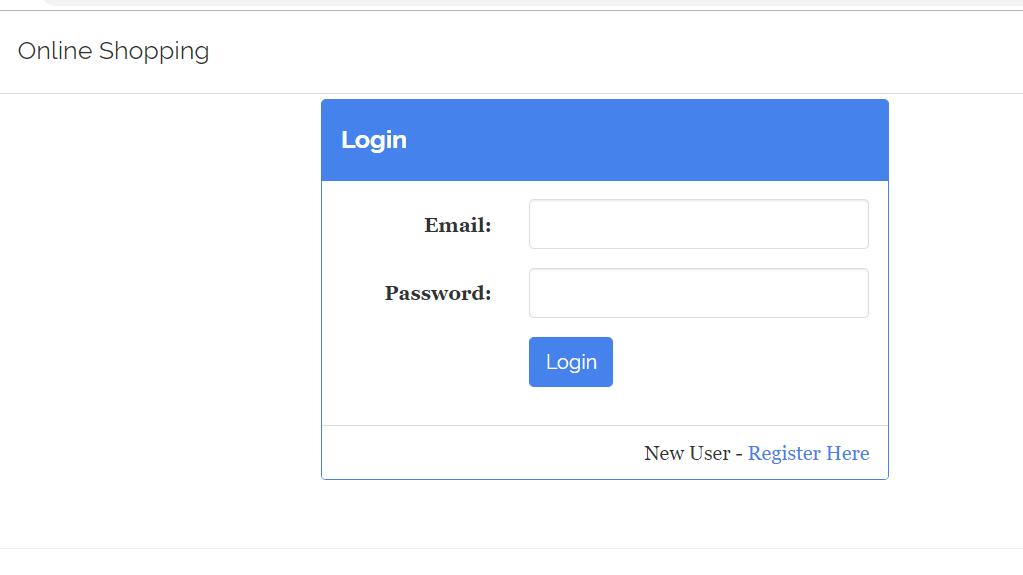


Figure 7.1: Login page

**Sign Up Page For User And Supplier**

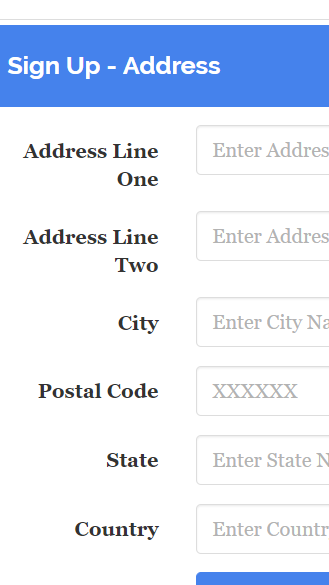
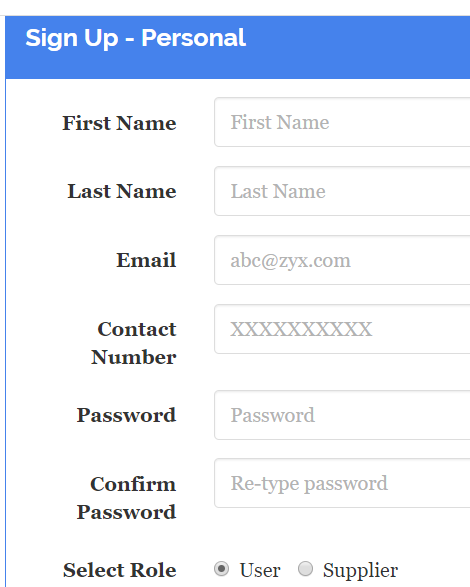
****

Figure 7.2: Sign Up Page

**Confirmation Detail**

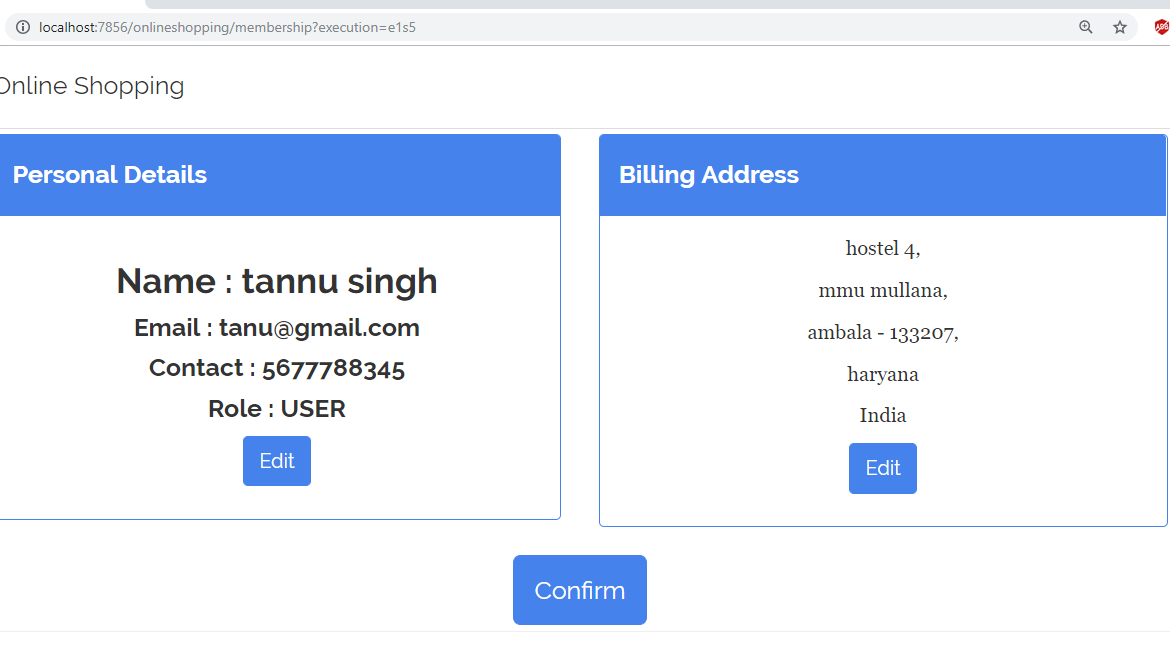


Figure 7.3: Confirmation of sign up detail

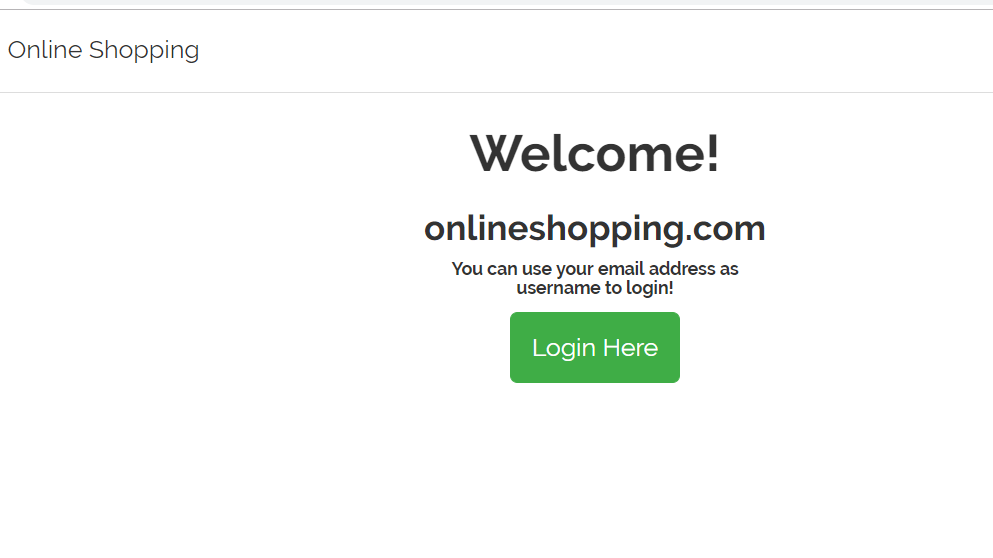
\

Figure 7.4: Welcome Page

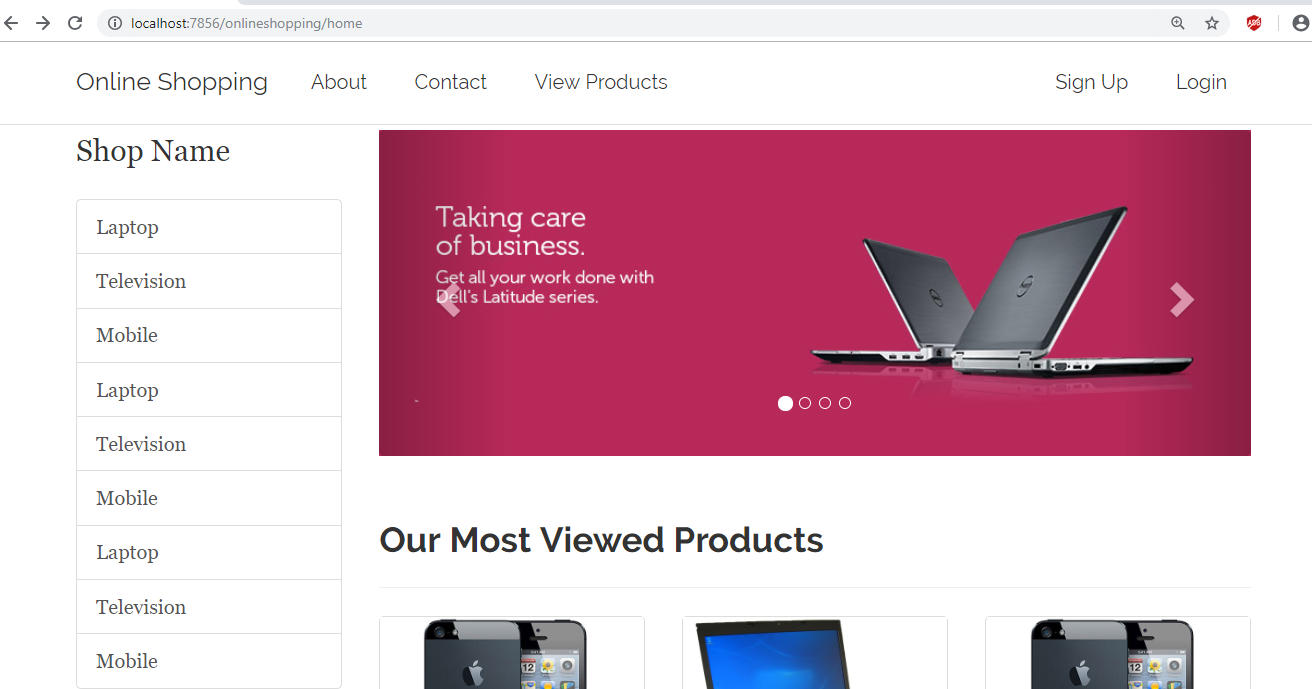


Figure 7.5: Home Page

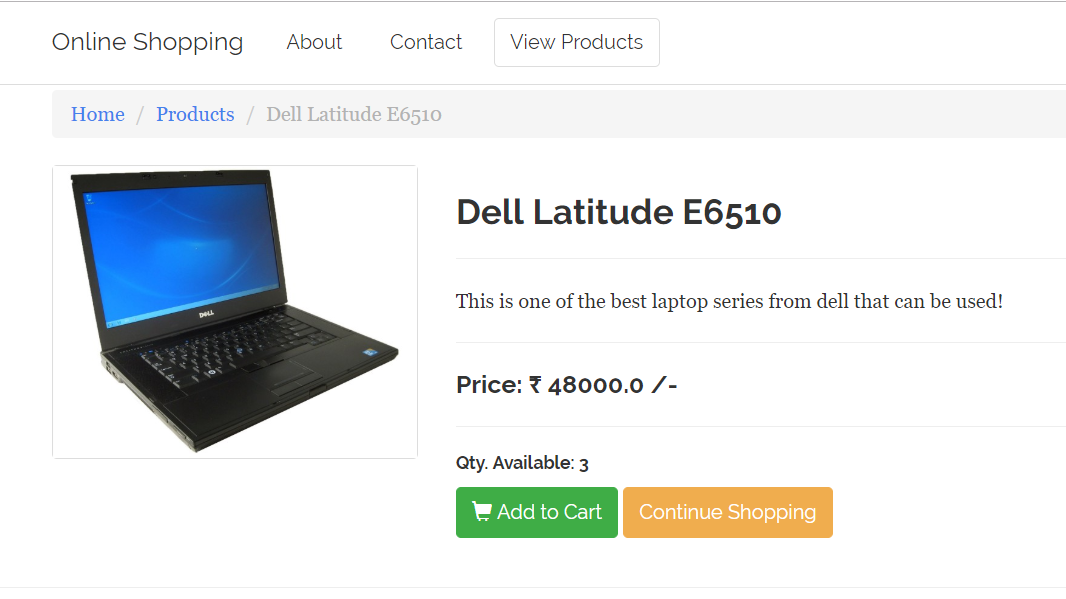


Figure 7.6: Product Detail

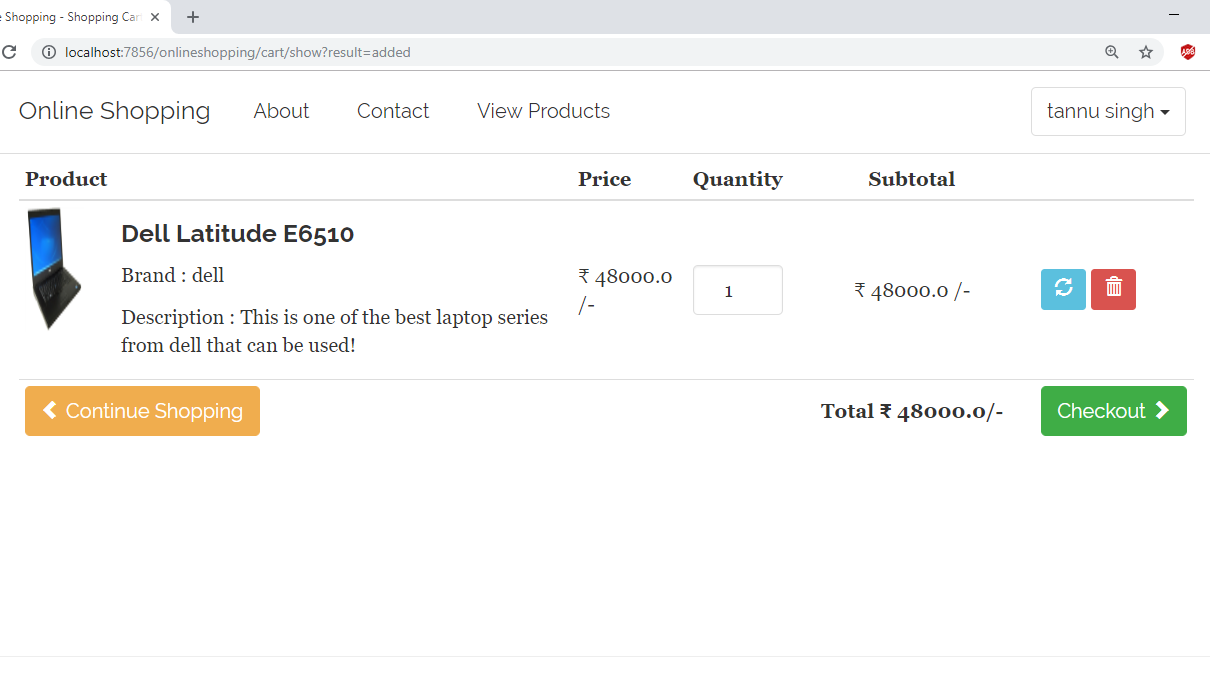


Figure 7.7: Product in Cart

# CHAPTER – 8

**TESTING AND IMPLEMENTATION**

**8.1 Testing**

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and code generation. The increasing visibility of software as a system element and the attendant “costs” associated with a software failure are motivating forces for well planned through testing.

Once source code has been generated, software must be tested to uncover as many errors as possible before delivery to customer. The goal is to design a series of test cases that online shopping system project report have a high likelihood of finding errors but how? That online shopping system project report’ where software testing techniques enter the pictures.

**8.1.1 Testing Objectives**

* Testing is a process of executing a program with the intent of finding an error.
* A good test case is one that online shopping system project report has a high probability of finding an as-yet-undiscovered error.
* A successful test is that online shopping system project report uncovers an as-yet- undiscovered error.

**8.1.2 Testing Principle**

* All tests should be traceable to customer requirement.
* Tests should be planned long before testing begins.
* The Pareto principle applies to software testing.
* Exhaustive testing is not possible.
* To be most effective, an independent third party should conduct testing.

"Software testing involves executing an implementation of the software with test data and examining the outputs of the software and its operational behavior to check that online shopping system project report it is performing as required. Testing is a dynamic technique of verification and validation because it works with an executable representation of the system

**8.1.3 Unit Testing**

Unit testing focuses verification effort on the smallest unit of software design-the software component or module. Using the component – level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and uncovered errors is limited by the constrained scope established for unit testing. The unit test is white-box oriented and the step can be conducted in parallel for multiple components.

**Login Module:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr. No** | **Test Case Description** | **Input** | **Expected Behavior** | **Observed behavior** | **Test Result** |
| 1 | Can ID field be Null? | Null ID | ID cannot be NULL | Warning msg "ID can't be NULL" | Success |
| 2 | Can password be Null? | Null pass­word | Password Can't be NULL | Warning msg "password can't be Null" | Success |
| 3 | Login button is working or not? | Button pressed | Perform login processing | Call proxy Inbox frame | Success |
| 4. | Is Login Frame displaying properly? | Invoke Login Frame | All text fields are displayed and are properly aligned | Little alignment problem | Success |

Table 5: Login module table

**8.1.4 Integration Testing**

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that online shopping system project report has been dictated by design. Incremental integration is the antithesis of the big bang approach. The program is constructed and tested in small increments, where errors are easier to isolate and correct, interfaces are more likely to be tested completely, and a systematic test approach may be applied.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr No** | **Test Case Description** | **Input** | **Expected Behavior** | **Observed behavior** | **Test Result** |
| 1 | Is new User created? | Login ID +personal information | User should be created and personal information should be stored in Database | User named ID is created. | Success |
| 2 | Is Database Connection establishing? | Connection object is created | Connection establishes | No error during connection was found | Success |
| 3 | Is able to match Login ID & Password | Login Id +Pass-word | Proper matching | Matching done | Success |
| 4 | Does status of user changes to 'Logout status’? | Logout cmd | Status should change. | No change in  status | Success |

Figure 6: Register, Login and Logout Module Integrated table

**8.1.5 System Testing**

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that online shopping system project report system elements have been properly integrated and perform allocated functions.

**8.1.6 OPTIMIZATION POINTS**

The software will work efficiently and speedily when the following conditions will be satisfied:

* The server should be of high configuration.
* The client machine has larger RAM.
* Adequate free space on the client’s hard disk.
* The user waits until he gets the home page properly.
* No access to the database for writing, deleting or updating by any other except the authority.

**Salient Features of the System**

* The software is completely menu driven.
* The data entry screens are completely user friendly.
* All editing features and navigation from one field to another, one web page to another, etc is possible.
* Exit from any web page is possible.
* Validation checks have been incorporated in each web page at the appropriate fields.
* Database has been secured by means of password protection.
* Authorization is necessary for all the internal users of the site.

# CHAPTER – 9

**FUTURE SCOPE**

**9.1 Limitations and Future Development**

There are some limitations for the current system to which solutions can be provided as a future development:

1. The system is not configured for multi- users at this online shopping system project time. The concept of *transaction* can be used to achieve this online shopping system project.
2. The Website is not accessible to everyone. It can be deployed on a web server so that online shopping system project report everybody who is connected to the Internet can use it.
3. Credit Card validation is not done. Third party proprietary software can be used for validation check.

**As for other future developments, the following can be done**:

1. The Administrator of the web site can be given more functionalities, like looking at a specific customer’s profile, the books that online shopping system project report have to be reordered, etc.
2. Multiple Shopping carts can be allowed.

**Conclusion**

The Internet has become a major resource in modern business, thus electronic shopping has gained significance not only from the entrepreneur’s but also from the customer’s point of view. For the entrepreneur, electronic shopping generates new business opportunities and for the customer, it makes comparative shopping possible. As per a survey, most consumers of online stores are impulsive and usually make a decision to stay on a site within the first few seconds. “Website design is like a shop interior. If the

shop looks poor or like hundreds of other shops the customer is most likely to skip to the other site. Hence we have designed the project to provide the user with easy navigation, retrieval of data and necessary feedback as much as possible.

In this online shopping system project project, the user is provided with an e-commerce web site that online shopping system project report can be used

to buy books online. To implement this online shopping system project as a web application we used JSP as the

Technology. JSP has several advantages such as enhanced performance, scalability, built- in security and simplicity. To build any web application using JSP we need a Programming language such as Java and JSP so on. was the language used to build this online shopping system project application. For the client browser to connect to the JSP engine we used Tomcat web server.

JSP uses JDBC to interact with the database as it provides in-memory caching that online shopping system project report eliminates the need to contact the database server frequently and it can easily deploy

and maintain an JSP application. Oracle was used as back-end database since it is one of the most popular commercial databases, and it provides fast data access, easy installation and simplicity.

A good shopping cart design must be accompanied with user-friendly shopping cart application logic. It should be convenient for the customer to view the contents of their cart and to be able to remove or add items to their cart. The shopping cart application described in this online shopping system project project provides a number of features that online shopping system project report are designed to make the customer more comfortable.

# 

**REFERENCE**

**Books & Authors**

* Naughton Schildt, Complete Reference Java 2, Third Edition, TMG.
* Ivor Horton, Beginning Java 2, Wrox Publications.
* Professional Java Server Programming J2EE 1.3 Edition, APress.
* Oracle 9i: The Complete Reference, Kevin Loney & George Koch, Oracle Press.
* Software Engineering, Pressman.

# APPENDIX: A

**SOURCE CODE**

package net.kzn.shoppingbackend.config;

import java.util.Properties;

import javax.sql.DataSource;

import org.apache.commons.dbcp2.BasicDataSource;

import org.hibernate.SessionFactory;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.core.annotation.Order;

import org.springframework.orm.hibernate5.HibernateTransactionManager;

import org.springframework.orm.hibernate5.LocalSessionFactoryBuilder;

import org.springframework.transaction.annotation.EnableTransactionManagement;

@Configuration

@ComponentScan(basePackages={"net.kzn.shoppingbackend.dto"})

@EnableTransactionManagement

public class HibernateConfig {

// Change the below based on the DBMS you choose

private final static String DATABASE\_URL = "jdbc:h2:tcp://localhost/~/onlineshopping";

private final static String DATABASE\_DRIVER = "org.h2.Driver";

private final static String DATABASE\_DIALECT = "org.hibernate.dialect.H2Dialect";

private final static String DATABASE\_USERNAME = "sa";

private final static String DATABASE\_PASSWORD = "";

// dataSource bean will be available

@Bean("dataSource")

public DataSource getDataSource() {

BasicDataSource dataSource = new BasicDataSource();

// Providing the database connection information

dataSource.setDriverClassName(DATABASE\_DRIVER);

dataSource.setUrl(DATABASE\_URL);

dataSource.setUsername(DATABASE\_USERNAME);

dataSource.setPassword(DATABASE\_PASSWORD);

return dataSource;

}

// sessionFactory bean will be available

@Bean

public SessionFactory getSessionFactory(DataSource dataSource) {

LocalSessionFactoryBuilder builder = new LocalSessionFactoryBuilder(dataSource);

builder.addProperties(getHibernateProperties());

builder.scanPackages("net.kzn.shoppingbackend.dto");

return builder.buildSessionFactory();

}

// All the hibernate properties will be returned in this method

private Properties getHibernateProperties() {

Properties properties = new Properties();

properties.put("hibernate.dialect", DATABASE\_DIALECT);

properties.put("hibernate.show\_sql", "true");

properties.put("hibernate.format\_sql", "true");

//properties.put("hibernate.hbm2ddl.auto", "create");

return properties;

}

// transactionManager bean

@Bean

public HibernateTransactionManager getTransactionManager(SessionFactory sessionFactory) {

HibernateTransactionManager transactionManager = new HibernateTransactionManager(sessionFactory);

return transactionManager;

}

}

CARTLINE DAO

package net.kzn.shoppingbackend.config;

import java.util.Properties;

import javax.sql.DataSource;

import org.apache.commons.dbcp2.BasicDataSource;

import org.hibernate.SessionFactory;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.core.annotation.Order;

import org.springframework.orm.hibernate5.HibernateTransactionManager;

import org.springframework.orm.hibernate5.LocalSessionFactoryBuilder;

import org.springframework.transaction.annotation.EnableTransactionManagement;

@Configuration

@ComponentScan(basePackages={"net.kzn.shoppingbackend.dto"})

@EnableTransactionManagement

public class HibernateConfig {

// Change the below based on the DBMS you choose

private final static String DATABASE\_URL = "jdbc:h2:tcp://localhost/~/onlineshopping";

private final static String DATABASE\_DRIVER = "org.h2.Driver";

private final static String DATABASE\_DIALECT = "org.hibernate.dialect.H2Dialect";

private final static String DATABASE\_USERNAME = "sa";

private final static String DATABASE\_PASSWORD = "";

// dataSource bean will be available

@Bean("dataSource")

public DataSource getDataSource() {

BasicDataSource dataSource = new BasicDataSource();

// Providing the database connection information

dataSource.setDriverClassName(DATABASE\_DRIVER);

dataSource.setUrl(DATABASE\_URL);

dataSource.setUsername(DATABASE\_USERNAME);

dataSource.setPassword(DATABASE\_PASSWORD);

return dataSource;

}

// sessionFactory bean will be available

@Bean

public SessionFactory getSessionFactory(DataSource dataSource) {

LocalSessionFactoryBuilder builder = new LocalSessionFactoryBuilder(dataSource);

builder.addProperties(getHibernateProperties());

builder.scanPackages("net.kzn.shoppingbackend.dto");

return builder.buildSessionFactory();

}

// All the hibernate properties will be returned in this method

private Properties getHibernateProperties() {

Properties properties = new Properties();

properties.put("hibernate.dialect", DATABASE\_DIALECT);

properties.put("hibernate.show\_sql", "true");

properties.put("hibernate.format\_sql", "true");

//properties.put("hibernate.hbm2ddl.auto", "create");

return properties;

}

// transactionManager bean

@Bean

public HibernateTransactionManager getTransactionManager(SessionFactory sessionFactory) {

HibernateTransactionManager transactionManager = new HibernateTransactionManager(sessionFactory);

return transactionManager;

}

}

package net.kzn.shoppingbackend.daoimpl;

import java.util.List;

import org.hibernate.SessionFactory;

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

import org.springframework.stereotype.Repository;

import org.springframework.transaction.annotation.Transactional;

import net.kzn.shoppingbackend.dao.CartLineDAO;

import net.kzn.shoppingbackend.dto.Cart;

import net.kzn.shoppingbackend.dto.CartLine;

import net.kzn.shoppingbackend.dto.OrderDetail;

@Repository("cartLineDAO")

@Transactional

public class CartLineDAOImpl implements CartLineDAO {

@Autowired

private SessionFactory sessionFactory;

@Override

public CartLine getByCartAndProduct(int cartId, int productId) {

String query = "FROM CartLine WHERE cartId = :cartId AND product.id = :productId";

try {

return sessionFactory.getCurrentSession()

.createQuery(query,CartLine.class) .setParameter("cartId", cartId)

.setParameter("productId", productId)

.getSingleResult();

}catch(Exception ex) {

return null;

}

}

@Override

public boolean add(CartLine cartLine) {

try {

sessionFactory.getCurrentSession().persist(cartLine);

return true;

}

catch(Exception ex) {

ex.printStackTrace();

return false;

}

}

@Override

public boolean update(CartLine cartLine) {

try {

sessionFactory.getCurrentSession().update(cartLine);

return true;

}

catch(Exception ex) {

ex.printStackTrace();

return false;

}

}

@Override

public boolean remove(CartLine cartLine) {

try {

sessionFactory.getCurrentSession().delete(cartLine);

return true;

}catch(Exception ex) {

return false;

}

}

@Override

public List<CartLine> list(int cartId) {

String query = "FROM CartLine WHERE cartId = :cartId";

return sessionFactory.getCurrentSession()

.createQuery(query, CartLine.class)

.setParameter("cartId", cartId)

.getResultList();

}

@Override

public CartLine get(int id) {

return sessionFactory.getCurrentSession().get(CartLine.class, Integer.valueOf(id));

}

@Override

public boolean updateCart(Cart cart) {

try {

sessionFactory.getCurrentSession().update(cart);

return true;

}

catch(Exception ex) {

return false;

}

}

@Override

public List<CartLine> listAvailable(int cartId) {

String query = "FROM CartLine WHERE cartId = :cartId AND available = :available";

return sessionFactory.getCurrentSession()

.createQuery(query, CartLine.class)

.setParameter("cartId", cartId)

.setParameter("available", true)

.getResultList();

}

@Override

public boolean addOrderDetail(OrderDetail orderDetail) {

try {

sessionFactory.getCurrentSession().persist(orderDetail);

return true;

}

catch(Exception ex) {

return false;

}

}

}

CATEGORY DAO

package net.kzn.shoppingbackend.dao;

import java.util.List;

import net.kzn.shoppingbackend.dto.Category;

public interface CategoryDAO {

Category get(int id);

List<Category> list();

boolean add(Category category);

boolean update(Category category);

boolean delete(Category category);

}

package net.kzn.shoppingbackend.daoimpl;

import java.util.List;

import org.hibernate.SessionFactory;

import org.hibernate.query.Query;

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

import org.springframework.stereotype.Repository;

import org.springframework.transaction.annotation.Transactional;

import net.kzn.shoppingbackend.dao.CategoryDAO;

import net.kzn.shoppingbackend.dto.Category;

@Repository("categoryDAO")

@Transactional

public class CategoryDAOImpl implements CategoryDAO {

@Autowired

private SessionFactory sessionFactory;

@Override

public List<Category> list() {

String selectActiveCategory = "FROM Category WHERE active = :active";

Query query = sessionFactory.getCurrentSession().createQuery(selectActiveCategory);

query.setParameter("active", true);

return query.getResultList();

}

/\*

\* Getting single category based on id

\*/

@Override

public Category get(int id) {

return sessionFactory.getCurrentSession().get(Category.class, Integer.valueOf(id));

}

@Override

public boolean add(Category category) {

try {

// add the category to the database table

sessionFactory.getCurrentSession().persist(category);

return true;

} catch (Exception ex) {

ex.printStackTrace();

return false;

}

}

/\*

\* Updating a single category

\*/

@Override

public boolean update(Category category) {

try {

// add the category to the database table

sessionFactory.getCurrentSession().update(category);

return true;

} catch (Exception ex) {

ex.printStackTrace();

return false;

}

}

@Override

public boolean delete(Category category) {

category.setActive(false);

try {

// add the category to the database table

sessionFactory.getCurrentSession().update(category);

return true;

} catch (Exception ex) {

ex.printStackTrace();

return false;

}

}

}

PRODUCT DAO

package net.kzn.shoppingbackend.dao;

import java.util.List;

import net.kzn.shoppingbackend.dto.Product;

public interface ProductDAO {

Product get(int productId);

List<Product> list();

boolean add(Product product);

boolean update(Product product);

boolean delete(Product product);

List<Product> getProductsByParam(String param, int count);

// business methods

List<Product> listActiveProducts();

List<Product> listActiveProductsByCategory(int categoryId);

List<Product> getLatestActiveProducts(int count);

}

package net.kzn.shoppingbackend.daoimpl;

import java.util.List;

import org.hibernate.SessionFactory;

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

import org.springframework.stereotype.Repository;

import org.springframework.transaction.annotation.Transactional;

import net.kzn.shoppingbackend.dao.ProductDAO;

import net.kzn.shoppingbackend.dto.Product;

@Repository("productDAO")

@Transactional

public class ProductDAOImpl implements ProductDAO {

@Autowired

private SessionFactory sessionFactory;

/\*

\* SINGLE

\* \*/

@Override

public Product get(int productId) {

try {

return sessionFactory

.getCurrentSession()

.get(Product.class,Integer.valueOf(productId));

}

catch(Exception ex) {

ex.printStackTrace();

}

return null;

}

/\*

\* LIST

\* \*/

@Override

public List<Product> list() {

return sessionFactory

.getCurrentSession()

.createQuery("FROM Product" , Product.class)

.getResultList();

}

/\*

\* INSERT

\* \*/

@Override

public boolean add(Product product) {

try {

sessionFactory

.getCurrentSession()

.persist(product);

return true;

}

catch(Exception ex) {

ex.printStackTrace();

}

return false;

}

/\*

\* UPDATE

\* \*/

@Override

public boolean update(Product product) {

try {

sessionFactory

.getCurrentSession()

.update(product);

return true;

}

catch(Exception ex) {

ex.printStackTrace();

}

return false;

}

/\*

\* DELETE

\* \*/

@Override

public boolean delete(Product product) {

try {

product.setActive(false);

// call the update method

return this.update(product);

}

catch(Exception ex) {

ex.printStackTrace();

}

return false;

}

@Override

public List<Product> listActiveProducts() {

String selectActiveProducts = "FROM Product WHERE active = :active";

return sessionFactory

.getCurrentSession()

.createQuery(selectActiveProducts, Product.class)

.setParameter("active", true)

.getResultList();

}

@Override

public List<Product> listActiveProductsByCategory(int categoryId) {

String selectActiveProductsByCategory = "FROM Product WHERE active = :active AND categoryId = :categoryId";

return sessionFactory

.getCurrentSession()

.createQuery(selectActiveProductsByCategory, Product.class)

.setParameter("active", true)

.setParameter("categoryId",categoryId)

.getResultList();

}

@Override

public List<Product> getLatestActiveProducts(int count) {

return sessionFactory

.getCurrentSession()

.createQuery("FROM Product WHERE active = :active ORDER BY id", Product.class)

.setParameter("active", true)

.setFirstResult(0)

.setMaxResults(count)

.getResultList();

}

@Override

public List<Product> getProductsByParam(String param, int count) {

String query = "FROM Product WHERE active = true ORDER BY " + param + " DESC";

return sessionFactory

.getCurrentSession()

.createQuery(query,Product.class)

.setFirstResult(0)

.setMaxResults(count)

.getResultList();

}

}

USER DAO

package net.kzn.shoppingbackend.dao;

import java.util.List;

import net.kzn.shoppingbackend.dto.Address;

import net.kzn.shoppingbackend.dto.Cart;

import net.kzn.shoppingbackend.dto.User;

public interface UserDAO {

// user related operation

User getByEmail(String email);

User get(int id);

boolean add(User user);

// adding and updating a new address

Address getAddress(int addressId);

boolean addAddress(Address address);

boolean updateAddress(Address address);

Address getBillingAddress(int userId);

List<Address> listShippingAddresses(int userId);

}

package net.kzn.shoppingbackend.daoimpl;

import java.util.List;

import org.hibernate.SessionFactory;

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

import org.springframework.stereotype.Repository;

import org.springframework.transaction.annotation.Transactional;

import net.kzn.shoppingbackend.dao.UserDAO;

import net.kzn.shoppingbackend.dto.Address;

import net.kzn.shoppingbackend.dto.User;

@Repository("userDAO")

@Transactional

public class UserDAOImpl implements UserDAO {

@Autowired

private SessionFactory sessionFactory;

@Override

public User getByEmail(String email) {

String selectQuery = "FROM User WHERE email = :email";

try {

return sessionFactory

.getCurrentSession()

.createQuery(selectQuery,User.class)

.setParameter("email",email)

.getSingleResult();

}

catch(Exception ex) {

return null;

}

}

@Override

public boolean add(User user) {

try {

sessionFactory.getCurrentSession().persist(user);

return true;

}

catch(Exception ex) {

return false;

}

}

@Override

public boolean addAddress(Address address) {

try {

// will look for this code later and why we need to change it

sessionFactory.getCurrentSession().persist(address);

return true;

}

catch(Exception ex) {

return false;

}

}

@Override

public boolean updateAddress(Address address) {

try {

sessionFactory.getCurrentSession().update(address);

return true;

}

catch(Exception ex) {

return false;

}

}

@Override

public List<Address> listShippingAddresses(int userId) {

String selectQuery = "FROM Address WHERE userId = :userId AND shipping = :isShipping ORDER BY id DESC";

return sessionFactory

.getCurrentSession()

.createQuery(selectQuery,Address.class)

.setParameter("userId", userId)

.setParameter("isShipping", true)

.getResultList();

}

@Override

public Address getBillingAddress(int userId) {

String selectQuery = "FROM Address WHERE userId = :userId AND billing = :isBilling";

try{

return sessionFactory

.getCurrentSession()

.createQuery(selectQuery,Address.class)

.setParameter("userId", userId)

.setParameter("isBilling", true)

.getSingleResult();

}

catch(Exception ex) {

return null;

}

}

@Override

public User get(int id) {

try {

return sessionFactory.getCurrentSession().get(User.class, id);

}

catch(Exception ex) {

System.out.println(ex.getMessage());

return null;

}

}

@Override

public Address getAddress(int addressId) {

try {

return sessionFactory.getCurrentSession().get(Address.class, addressId);

}

catch(Exception ex) {

System.out.println(ex.getMessage());

return null;

}

}

ADDRESS

package net.kzn.shoppingbackend.dto;

import java.io.Serializable;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import org.hibernate.validator.constraints.NotBlank;

@Entity

public class Address implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@NotBlank(message = "Please enter address line one!")

@Column(name = "address\_line\_one")

private String addressLineOne;

@NotBlank(message = "Please enter address line two!")

@Column(name = "address\_line\_two")

private String addressLineTwo;

@NotBlank(message = "Please enter City!")

private String city;

@NotBlank(message = "Please enter State!")

private String state;

@NotBlank(message = "Please enter country!")

private String country;

@Column(name ="postal\_code")

@NotBlank(message = "Please enter Postal Code!")

private String postalCode;

@Column(name="is\_shipping")

private boolean shipping;

@Column(name="is\_billing")

private boolean billing;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getAddressLineOne() {

return addressLineOne;

}

public void setAddressLineOne(String addressLineOne) {

this.addressLineOne = addressLineOne;

}

public String getAddressLineTwo() {

return addressLineTwo;

}

public void setAddressLineTwo(String addressLineTwo) {

this.addressLineTwo = addressLineTwo;

}

public String getCity() {

return city;

}

public void setCity(String city) {

this.city = city;

}

public String getState() {

return state;

}

public void setState(String state) {

this.state = state;

}

public String getCountry() {

return country;

}

public void setCountry(String country) {

this.country = country;

}

public String getPostalCode() {

return postalCode;

}

public void setPostalCode(String postalCode) {

this.postalCode = postalCode;

}

public boolean isBilling() {

return billing;

}

public void setBilling(boolean billing) {

this.billing = billing;

}

@Override

public String toString() {

return "Address [id=" + id + ", addressLineOne=" + addressLineOne + ", addressLineTwo=" + addressLineTwo

+ ", city=" + city + ", state=" + state + ", country=" + country + ", postalCode=" + postalCode

+ ", billing=" + billing + "]";

}

@Column(name = "user\_id")

private int userId;

public boolean isShipping() {

return shipping;

}

public void setShipping(boolean shipping) {

this.shipping = shipping;

}

public int getUserId() {

return userId;

}

public void setUserId(int userId) {

this.userId = userId;

}

}

CART

package net.kzn.shoppingbackend.dto;

import java.io.Serializable;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToOne;

@Entity

public class Cart implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name = "grand\_total")

private double grandTotal;

@Column(name = "cart\_lines")

private int cartLines;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public double getGrandTotal() {

return grandTotal;

}

public void setGrandTotal(double grandTotal) {

this.grandTotal = grandTotal;

}

public int getCartLines() {

return cartLines;

}

public void setCartLines(int cartLines) {

this.cartLines = cartLines;

}

@Override

public String toString() {

return "Cart [id=" + id + ", grandTotal=" + grandTotal + ", cartLines=" + cartLines + "]";

}

// linking the cart with a user

@OneToOne

private User user;

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

}

ORDER DETAIL

package net.kzn.shoppingbackend.dto;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import javax.persistence.OneToMany;

import javax.persistence.Table;

@Entity

@Table(name = "order\_detail")

public class OrderDetail implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@ManyToOne

@JoinColumn(name = "user\_id")

private User user;

@Column(name = "order\_total")

private double orderTotal;

@ManyToOne

private Address shipping;

@ManyToOne

private Address billing;

@OneToMany(mappedBy="orderDetail", fetch = FetchType.EAGER, cascade = CascadeType.ALL)

private List<OrderItem> orderItems = new ArrayList<>();

@Column(name = "order\_count")

private int orderCount;

@Column(name="order\_date")

private Date orderDate;

public Date getOrderDate() {

return orderDate;

}

public void setOrderDate(Date orderDate) {

this.orderDate = orderDate;

}

public int getOrderCount() {

return orderCount;

}

public void setOrderCount(int orderCount) {

this.orderCount = orderCount;

}

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public double getOrderTotal() {

return orderTotal;

}

public void setOrderTotal(double orderTotal) {

this.orderTotal = orderTotal;

}

public Address getShipping() {

return shipping;

}

public void setShipping(Address shipping) {

this.shipping = shipping;

}

public Address getBilling() {

return billing;

}

public void setBilling(Address billing) {

this.billing = billing;

}

public List<OrderItem> getOrderItems() {

return orderItems;

}

public void setOrderItems(List<OrderItem> orderItems) {

this.orderItems = orderItems;

}

}

USER

package net.kzn.shoppingbackend.dto;

import java.io.Serializable;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToOne;

import javax.persistence.Table;

import javax.persistence.Transient;

import org.hibernate.validator.constraints.NotBlank;

@Entity

@Table(name = "user\_detail")

public class User implements Serializable{

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@NotBlank(message = "Please enter first name!")

@Column(name = "first\_name")

private String firstName;

@NotBlank(message = "Please enter last name!")

@Column(name = "last\_name")

private String lastName;

@NotBlank(message = "Please enter email address!")

private String email;

@NotBlank(message = "Please enter contact number!")

@Column(name = "contact\_number")

private String contactNumber;

private String role;

@NotBlank(message = "Please enter password!")

private String password;

private boolean enabled = true;

@Transient

private String confirmPassword;

public String getConfirmPassword() {

return confirmPassword;

}

public void setConfirmPassword(String confirmPassword) {

this.confirmPassword = confirmPassword;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getContactNumber() {

return contactNumber;

}

public void setContactNumber(String contactNumber) {

this.contactNumber = contactNumber;

}

public String getRole() {

return role;

}

public void setRole(String role) {

this.role = role;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public boolean isEnabled() {

return enabled;

}

public void setEnabled(boolean enabled) {

this.enabled = enabled;

}

@Override

public String toString() {

return "User [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ", email=" + email

+ ", contactNumber=" + contactNumber + ", role=" + role + ", password=" + password + ", enabled="

+ enabled + "]";

}

@OneToOne(mappedBy="user", cascade = CascadeType.ALL, fetch = FetchType.EAGER)

private Cart cart;

public Cart getCart() {

return cart;

}

public void setCart(Cart cart) {

this.cart = cart;

}

}

DATABASE QUERIES

**CREATE** **TABLE** category (

id IDENTITY,

name **VARCHAR**(50),

description **VARCHAR**(255),

image\_url **VARCHAR**(50),

is\_active **BOOLEAN**,

**CONSTRAINT** pk\_category\_id **PRIMARY** **KEY** (id)

);

**CREATE** **TABLE** user\_detail (

id IDENTITY,

first\_name **VARCHAR**(50),

last\_name **VARCHAR**(50),

role **VARCHAR**(50),

enabled **BOOLEAN**,

password **VARCHAR**(60),

email **VARCHAR**(100),

contact\_number **VARCHAR**(15),

**CONSTRAINT** pk\_user\_id **PRIMARY** **KEY**(id)

);

**CREATE** **TABLE** product (

id IDENTITY,

code **VARCHAR**(20),

name **VARCHAR**(50),

brand **VARCHAR**(50),

description **VARCHAR**(255),

unit\_price **DECIMAL**(10,2),

quantity **INT**,

is\_active **BOOLEAN**,

category\_id **INT**,

supplier\_id **INT**,

purchases **INT** **DEFAULT** 0,

views **INT** **DEFAULT** 0,

**CONSTRAINT** pk\_product\_id **PRIMARY** **KEY** (id),

**CONSTRAINT** fk\_product\_category\_id **FOREIGN** **KEY** (category\_id) **REFERENCES** category (id),

**CONSTRAINT** fk\_product\_supplier\_id **FOREIGN** **KEY** (supplier\_id) **REFERENCES** user\_detail(id),

);

-- the address table to store the user billing and shipping addresses

**CREATE** **TABLE** address (

id IDENTITY,

user\_id **int**,

address\_line\_one **VARCHAR**(100),

address\_line\_two **VARCHAR**(100),

city **VARCHAR**(20),

state **VARCHAR**(20),

country **VARCHAR**(20),

postal\_code **VARCHAR**(10),

is\_billing **BOOLEAN**,

is\_shipping **BOOLEAN**,

**CONSTRAINT** fk\_address\_user\_id **FOREIGN** **KEY** (user\_id ) **REFERENCES** user\_detail (id),

**CONSTRAINT** pk\_address\_id **PRIMARY** **KEY** (id)

);

-- the cart table to store the user cart top-level details

**CREATE** **TABLE** cart (

id IDENTITY,

user\_id **int**,

grand\_total **DECIMAL**(10,2),

cart\_lines **int**,

**CONSTRAINT** fk\_cart\_user\_id **FOREIGN** **KEY** (user\_id ) **REFERENCES** user\_detail (id),

**CONSTRAINT** pk\_cart\_id **PRIMARY** **KEY** (id)

);

-- the cart line table to store the cart details

**CREATE** **TABLE** cart\_line (

id IDENTITY,

cart\_id **int**,

total **DECIMAL**(10,2),

product\_id **int**,

product\_count **int**,

buying\_price **DECIMAL**(10,2),

is\_available **boolean**,

**CONSTRAINT** fk\_cartline\_product\_id **FOREIGN** **KEY** (product\_id ) **REFERENCES** product (id),

**CONSTRAINT** pk\_cartline\_id **PRIMARY** **KEY** (id)

);

-- the order detail table to store the order

**CREATE** **TABLE** order\_detail (

id IDENTITY,

user\_id **int**,

order\_total **DECIMAL**(10,2),

order\_count **int**,

shipping\_id **int**,

billing\_id **int**,

order\_date **date**,

**CONSTRAINT** fk\_order\_detail\_user\_id **FOREIGN** **KEY** (user\_id) **REFERENCES** user\_detail (id),

**CONSTRAINT** fk\_order\_detail\_shipping\_id **FOREIGN** **KEY** (shipping\_id) **REFERENCES** address (id),

**CONSTRAINT** fk\_order\_detail\_billing\_id **FOREIGN** **KEY** (billing\_id) **REFERENCES** address (id),

**CONSTRAINT** pk\_order\_detail\_id **PRIMARY** **KEY** (id)

);

-- the order item table to store order items

**CREATE** **TABLE** order\_item (

id IDENTITY,

order\_id **int**,

total **DECIMAL**(10,2),

product\_id **int**,

product\_count **int**,

buying\_price **DECIMAL**(10,2),

**CONSTRAINT** fk\_order\_item\_product\_id **FOREIGN** **KEY** (product\_id) **REFERENCES** product (id),

**CONSTRAINT** fk\_order\_item\_order\_id **FOREIGN** **KEY** (order\_id) **REFERENCES** order\_detail (id),

**CONSTRAINT** pk\_order\_item\_id **PRIMARY** **KEY** (id)

);

-- adding three categories

**INSERT** **INTO** category (name, description,image\_url,is\_active) **VALUES** ('Laptop', 'This is description for Laptop category!', 'CAT\_1.png', **true**);

**INSERT** **INTO** category (name, description,image\_url,is\_active) **VALUES** ('Television', 'This is description for Television category!', 'CAT\_2.png', **true**);

**INSERT** **INTO** category (name, description,image\_url,is\_active) **VALUES** ('Mobile', 'This is description for Mobile category!', 'CAT\_3.png', **true**);

-- adding three users

**INSERT** **INTO** user\_detail

(first\_name, last\_name, role, enabled, password, email, contact\_number)

**VALUES** ('Virat', 'Kohli', 'ADMIN', **true**, '$2a$06$ORtBskA2g5Wg0HDgRE5ZsOQNDHUZSdpJqJ2.PGXv0mKyEvLnKP7SW', 'vk@gmail.com', '8888888888');

**INSERT** **INTO** user\_detail

(first\_name, last\_name, role, enabled, password, email, contact\_number)

**VALUES** ('Ravindra', 'Jadeja', 'SUPPLIER', **true**, '$2a$06$bzYMivkRjSxTK2LPD8W4te6jjJa795OwJR1Of5n95myFsu3hgUnm6', 'rj@gmail.com', '9999999999');

**INSERT** **INTO** user\_detail

(first\_name, last\_name, role, enabled, password, email, contact\_number)

**VALUES** ('Ravichandra', 'Ashwin', 'SUPPLIER', **true**, '$2a$06$i1dLNlXj2uY.UBIb9kUcAOxCigGHUZRKBtpRlmNtL5xtgD6bcVNOK', 'ra@gmail.com', '7777777777');

**INSERT** **INTO** user\_detail

(first\_name, last\_name, role, enabled, password, email, contact\_number)

**VALUES** ('Khozema', 'Nullwala', 'USER', **true**, '$2a$06$4mvvyO0h7vnUiKV57IW3oudNEaKPpH1xVSdbie1k6Ni2jfjwwminq', 'kn@gmail.com', '7777777777');

-- adding five products

**INSERT** **INTO** product (code, name, brand, description, unit\_price, quantity, is\_active, category\_id, supplier\_id, purchases, views)

**VALUES** ('PRDABC123DEFX', 'iPhone 5s', 'apple', 'This is one of the best phone available in the market right now!', 18000, 5, **true**, 3, 2, 0, 0 );

**INSERT** **INTO** product (code, name, brand, description, unit\_price, quantity, is\_active, category\_id, supplier\_id, purchases, views)

**VALUES** ('PRDDEF123DEFX', 'Samsung s7', 'samsung', 'A smart phone by samsung!', 32000, 2, **true**, 3, 3, 0, 0 );

**INSERT** **INTO** product (code, name, brand, description, unit\_price, quantity, is\_active, category\_id, supplier\_id, purchases, views)

**VALUES** ('PRDPQR123WGTX', 'Google Pixel', 'google', 'This is one of the best android smart phone available in the market right now!', 57000, 5, **true**, 3, 2, 0, 0 );

**INSERT** **INTO** product (code, name, brand, description, unit\_price, quantity, is\_active, category\_id, supplier\_id, purchases, views)

**VALUES** ('PRDMNO123PQRX', ' Macbook Pro', 'apple', 'This is one of the best laptops available in the market right now!', 54000, 3, **true**, 1, 2, 0, 0 );

**INSERT** **INTO** product (code, name, brand, description, unit\_price, quantity, is\_active, category\_id, supplier\_id, purchases, views)

**VALUES** ('PRDABCXYZDEFX', 'Dell Latitude E6510', 'dell', 'This is one of the best laptop series from dell that can be used!', 48000, 5, **true**, 1, 3, 0, 0 );

-- adding a supplier correspondece address

**INSERT** **INTO** address( user\_id, address\_line\_one, address\_line\_two, city, state, country, postal\_code, is\_billing, is\_shipping)

**VALUES** (4, '102 Sabarmati Society, Mahatma Gandhi Road', 'Near Salt Lake, Gandhi Nagar', 'Ahmedabad', 'Gujarat', 'India', '111111', **true**, **false** );

-- adding a cart for testing

**INSERT** **INTO** cart (user\_id, grand\_total, cart\_lines) **VALUES** (4, 0, 0);

<!-- H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>1.4.196</version>

</dependency>

<!-- Hibernate Dependency -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>${hibernate.version}</version>

</dependency>

<!-- Database Connection Pooling -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-dbcp2</artifactId>

<version>2.1.1</version>

<exclusions>

<exclusion>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- Jackson -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>${jackson.version}</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-annotations</artifactId>

<version>${jackson.version}</version>

</dependency>

<!-- SLF4J Logging -->

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.1</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>jcl-over-slf4j</artifactId>

<version>1.7.24</version>

</dependency>

<!-- Hibernate Validator -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-validator</artifactId>

<version>5.4.1.Final</version>

</dependency>

<dependency>

<groupId>javax.validation</groupId>

<artifactId>validation-api</artifactId>

<version>1.1.0.Final</version>

</dependency>

</dependencies>

<build>

<!-- Updated for the latest version of JAVA -->

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

<web-app xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee*

*http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"*

version=*"3.1"*>

<display-name>Archetype Created Web Application</display-name>

<!-- Configuring front-controller -->

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- FOr throwing exception -->

<init-param>

<param-name>throwExceptionIfNoHandlerFound</param-name>

<param-value>true</param-value>

</init-param>

<multipart-config>

<max-file-size>20848820</max-file-size>

<max-request-size>418018841</max-request-size>

<file-size-threshold>1048576</file-size-threshold>

</multipart-config>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<!-- Configuration for spring security -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!-- Load spring security config file -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>

/WEB-INF/spring-security.xml

</param-value>

</context-param>

<!-- Spring Security filter -->

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

</web-app>

Model

**package** net.kzn.onlineshopping.model;

**import** java.io.Serializable;

**import** java.util.List;

**import** net.kzn.shoppingbackend.dto.Address;

**import** net.kzn.shoppingbackend.dto.Cart;

**import** net.kzn.shoppingbackend.dto.CartLine;

**import** net.kzn.shoppingbackend.dto.OrderDetail;

**import** net.kzn.shoppingbackend.dto.User;

**public** **class** CheckoutModel **implements** Serializable {

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** User user;

**private** Address shipping;

**private** Cart cart;

**private** List<CartLine> cartLines;

**private** OrderDetail orderDetail;

**private** **double** checkoutTotal;

**public** OrderDetail getOrderDetail() {

**return** orderDetail;

}

**public** **void** setOrderDetail(OrderDetail orderDetail) {

**this**.orderDetail = orderDetail;

}

**public** Cart getCart() {

**return** cart;

}

**public** **void** setCart(Cart cart) {

**this**.cart = cart;

}

**public** **double** getCheckoutTotal() {

**return** checkoutTotal;

}

**public** **void** setCheckoutTotal(**double** checkoutTotal) {

**this**.checkoutTotal = checkoutTotal;

}

**public** List<CartLine> getCartLines() {

**return** cartLines;

}

**public** **void** setCartLines(List<CartLine> cartLines) {

**this**.cartLines = cartLines;

}

**public** User getUser() {

**return** user;

}

**public** **void** setUser(User user) {

**this**.user = user;

}

**public** Address getShipping() {

**return** shipping;

}

**public** **void** setShipping(Address shipping) {

**this**.shipping = shipping;

}

}

package net.kzn.onlineshopping.model;

import java.io.Serializable;

import net.kzn.shoppingbackend.dto.Address;

import net.kzn.shoppingbackend.dto.User;

public class RegisterModel implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private User user;

private Address billing;

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public Address getBilling() {

return billing;

}

public void setBilling(Address billing) {

this.billing = billing;

}

}

package net.kzn.onlineshopping.model;

import java.io.Serializable;

import net.kzn.shoppingbackend.dto.Cart;

public class UserModel implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private int id;

private String fullName;

private String role;

public String getRole() {

return role;

}

public void setRole(String role) {

this.role = role;

}

private Cart cart;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

public Cart getCart() {

return cart;

}

public void setCart(Cart cart) {

this.cart = cart;

}

}

Service

package net.kzn.onlineshopping.service;

import java.util.List;

import javax.servlet.http.HttpSession;

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

import org.springframework.stereotype.Service;

import net.kzn.onlineshopping.model.UserModel;

import net.kzn.shoppingbackend.dao.CartLineDAO;

import net.kzn.shoppingbackend.dao.ProductDAO;

import net.kzn.shoppingbackend.dao.UserDAO;

import net.kzn.shoppingbackend.dto.Cart;

import net.kzn.shoppingbackend.dto.CartLine;

import net.kzn.shoppingbackend.dto.Product;

@Service("cartService")

public class CartService {

@Autowired

private CartLineDAO cartLineDAO;

@Autowired

private ProductDAO productDAO;

@Autowired

private HttpSession session;

public List<CartLine> getCartLines() {

return cartLineDAO.list(getCart().getId());

}

/\* to update the cart count \*/

public String manageCartLine(int cartLineId, int count) {

CartLine cartLine = cartLineDAO.get(cartLineId);

double oldTotal = cartLine.getTotal();

Product product = cartLine.getProduct();

// check if that much quantity is available or not

if(product.getQuantity() < count) {

return "result=unavailable";

}

// update the cart line

cartLine.setProductCount(count);

cartLine.setBuyingPrice(product.getUnitPrice());

cartLine.setTotal(product.getUnitPrice() \* count);

cartLineDAO.update(cartLine);

// update the cart

Cart cart = this.getCart();

cart.setGrandTotal(cart.getGrandTotal() - oldTotal + cartLine.getTotal());

cartLineDAO.updateCart(cart);

return "result=updated";

}

public String addCartLine(int productId) {

Cart cart = this.getCart();

String response = null;

CartLine cartLine = cartLineDAO.getByCartAndProduct(cart.getId(), productId);

if(cartLine==null) {

// add a new cartLine if a new product is getting added

cartLine = new CartLine();

Product product = productDAO.get(productId);

// transfer the product details to cartLine

cartLine.setCartId(cart.getId());

cartLine.setProduct(product);

cartLine.setProductCount(1);

cartLine.setBuyingPrice(product.getUnitPrice());

cartLine.setTotal(product.getUnitPrice());

// insert a new cartLine

cartLineDAO.add(cartLine);

// update the cart

cart.setGrandTotal(cart.getGrandTotal() + cartLine.getTotal());

cart.setCartLines(cart.getCartLines() + 1);

cartLineDAO.updateCart(cart);

response = "result=added";

}

else {

// check if the cartLine has been already reached to maximum count

if(cartLine.getProductCount() < 3) {

// call the manageCartLine method to increase the count

response = this.manageCartLine(cartLine.getId(), cartLine.getProductCount() + 1);

}

else {

response = "result=maximum";

}

}

return response;

}

private Cart getCart() {

return ((UserModel)session.getAttribute("userModel")).getCart();

}

public String removeCartLine(int cartLineId) {

CartLine cartLine = cartLineDAO.get(cartLineId);

// deduct the cart

// update the cart

Cart cart = this.getCart();

cart.setGrandTotal(cart.getGrandTotal() - cartLine.getTotal());

cart.setCartLines(cart.getCartLines() - 1);

cartLineDAO.updateCart(cart);

// remove the cartLine

cartLineDAO.remove(cartLine);

return "result=deleted";

}

public String validateCartLine() {

Cart cart = this.getCart();

List<CartLine> cartLines = cartLineDAO.list(cart.getId());

double grandTotal = 0.0;

int lineCount = 0;

String response = "result=success";

boolean changed = false;

Product product = null;

for(CartLine cartLine : cartLines) {

product = cartLine.getProduct();

changed = false;

// check if the product is active or not

// if it is not active make the availability of cartLine as false

if((!product.isActive() && product.getQuantity() == 0) && cartLine.isAvailable()) {

cartLine.setAvailable(false);

changed = true;

}

// check if the cartLine is not available

// check whether the product is active and has at least one quantity available

if((product.isActive() && product.getQuantity() > 0) && !(cartLine.isAvailable())) {

cartLine.setAvailable(true);

changed = true;

}

// check if the buying price of product has been changed

if(cartLine.getBuyingPrice() != product.getUnitPrice()) {

// set the buying price to the new price

cartLine.setBuyingPrice(product.getUnitPrice());

// calculate and set the new total

cartLine.setTotal(cartLine.getProductCount() \* product.getUnitPrice());

changed = true;

}

// check if that much quantity of product is available or not

if(cartLine.getProductCount() > product.getQuantity()) {

cartLine.setProductCount(product.getQuantity());

cartLine.setTotal(cartLine.getProductCount() \* product.getUnitPrice());

changed = true;

}

// changes has happened

if(changed) {

//update the cartLine

cartLineDAO.update(cartLine);

// set the result as modified

response = "result=modified";

}

grandTotal += cartLine.getTotal();

lineCount++;

}

cart.setCartLines(lineCount++);

cart.setGrandTotal(grandTotal);

cartLineDAO.updateCart(cart);

return response;

}

}

Util

package net.kzn.onlineshopping.util;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.URL;

import javax.servlet.http.HttpServletRequest;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.web.multipart.MultipartFile;

public class FileUtil {

private static final String ABS\_PATH = "E:/JAVAApp/online-shopping/onlineshopping/src/main/webapp/assets/images/";

private static String REAL\_PATH = null;

private static final Logger logger = LoggerFactory.getLogger(FileUtil.class);

public static boolean uploadFile(HttpServletRequest request, MultipartFile file, String code)

{

// get the real server path

REAL\_PATH = request.getSession().getServletContext().getRealPath("/assets/images/");

logger.info(REAL\_PATH);

// create the directories if it does not exist

if(!new File(REAL\_PATH).exists()) {

new File(REAL\_PATH).mkdirs();

}

if(!new File(ABS\_PATH).exists()) {

new File(ABS\_PATH).mkdirs();

}

try {

//transfer the file to both the location

file.transferTo(new File(REAL\_PATH + code + ".jpg"));

file.transferTo(new File(ABS\_PATH + code + ".jpg"));

}

catch(IOException ex) {

ex.printStackTrace();

}

return true;

}

public static void uploadNoImage(HttpServletRequest request, String code) {

// get the real server path

REAL\_PATH = request.getSession().getServletContext().getRealPath("/assets/images/");

String imageURL = "http://placehold.it/640X480?text=No Image";

String destinationServerFile = REAL\_PATH + code + ".jpg";

String destinationProjectFile = REAL\_PATH + code + ".jpg";

try {

URL url = new URL(imageURL);

try (

InputStream is = url.openStream();

OutputStream osREAL\_PATH = new FileOutputStream(destinationServerFile);

OutputStream osABS\_PATH = new FileOutputStream(destinationProjectFile);

){

byte[] b = new byte[2048];

int length;

while((length = is.read(b))!= -1) {

osREAL\_PATH.write(b, 0, length);

osABS\_PATH.write(b, 0, length);

}

}

}

catch(IOException ex) {

ex.printStackTrace();

}

}

}