THE DESIGN AND IMPLEMENTATION OF A SMALL TO MEDIUM RESTAURANT BUSINESS WEB APPLICATION

A graduate project submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

By

Edward Gerhardstein

May 2011
The graduate project of Edward Gerhardstein is approved:

_________________________________________  _______________________________________
John Noga, Ph.D.                          Date

_________________________________________  _______________________________________
Robert McIlhenny, Ph.D.                    Date

_________________________________________  _______________________________________
Jeff Wiegley, Ph.D., Chair                 Date

California State University, Northridge

ii
# Table of Contents

Signature page .......................................................... ii
Abstract .............................................................................. vi

1 Overview of Pizza Application ........................................... 1

2 Open Source Licenses Servers ........................................... 2
   2.1 Open Source License Definition ....................................... 2
   2.2 Ubuntu ........................................................................ 2
   2.3 Apache Tomcat .............................................................. 2
   2.4 MySQL ........................................................................ 4

3 Selected Concepts and Terminologies ................................. 6
   3.1 Model-View-Controller (MVC) .......................................... 6
   3.2 JavaScript ...................................................................... 7
   3.3 Ajax ........................................................................... 7
   3.4 XML ............................................................................ 7
   3.5 DTD ............................................................................ 7
   3.6 XML Schema .................................................................. 7
   3.7 CSS ........................................................................... 8

4 J2EE Concepts ................................................................... 9
   4.1 J2EE Overview .............................................................. 9
   4.2 JavaBean ...................................................................... 9
   4.3 Enterprise JavaBeans (EJB) .............................................. 9
   4.4 Other J2EE APIs and Technologies ................................. 9
   4.5 Servlets ....................................................................... 10
   4.6 JavaServer Pages (JSP) ................................................... 11
      4.6.1 Scriptlet ................................................................. 11

5 Apache Struts Framework .................................................. 13
   5.1 Apache Struts Overview ................................................ 13
   5.2 ActionServlet ............................................................... 13
   5.3 Struts Config ............................................................... 13

6 Pizza Application Overview ............................................... 15
   6.1 Design Layout ............................................................. 15
   6.2 Workflow ..................................................................... 15
   6.3 JSP Page formats - Index.jsp/Templates ......................... 17
   6.4 JSP Page Divisions ...................................................... 18

7 ClockIn/Clockout and Logon Functionality ......................... 21
   7.1 ClockIn/Clockout Functionality ...................................... 21
7.2 Logon Functionality ........................................... 21

8 Administrator Functionality ........................................ 24
  8.1 Administrator Functionality Description ...................... 24
  8.2 Product Administration ...................................... 24
    8.2.1 Size Maintenance ..................................... 26
    8.2.2 Category Maintenance .................................. 27
    8.2.3 Product Maintenance .................................. 28
    8.2.4 Crust Maintenance ..................................... 30
    8.2.5 Item Maintenance ...................................... 31
  8.3 Inventory Administration .................................... 33
    8.3.1 Inventory Maintenance .................................. 34
    8.3.2 Inventory Product Maintenance ...................... 35
    8.3.3 Inventory Item Maintenance ......................... 37
    8.3.4 Inventory Crust Maintenance ...................... 38
  8.4 Employee Maintenance ...................................... 40
  8.5 User Maintenance ........................................... 42

9 Management Functionality ......................................... 45
  9.1 Management Functionality Overview ....................... 45
  9.2 Schedule Management ...................................... 45
  9.3 Inventory Management ...................................... 47
  9.4 Inventory Delivery ......................................... 47
  9.5 Inventory Delivery ......................................... 48
  9.6 Report Management ......................................... 50
    9.6.1 Inventory Report ...................................... 50
    9.6.2 Driver Report .......................................... 51
    9.6.3 Employee Report ...................................... 52
    9.6.4 Product Report ......................................... 52
    9.6.5 Order Type Report .................................... 53

10 Order Functionality ................................................ 54
  10.1 Order Functionality Overview ............................. 54
  10.2 Order Entry Page .......................................... 55
  10.3 Order Item Type ........................................... 57
  10.4 Order Item .................................................. 57
  10.5 Order Sum ................................................... 59
  10.6 Order Change ............................................... 60

11 Dispatch /Cashier Functionality ................................. 63
  11.1 Dispatch /Cashier Functionality Overview ................ 63
  11.2 Cashier Main ............................................... 63
  11.3 Dispatch Main ............................................... 64
    11.3.1 Dispatch Order ....................................... 65
    11.3.2 Driver Bank ........................................... 65
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.3.3 Driver In</td>
<td>65</td>
</tr>
<tr>
<td>11.3.4 Cashout Driver</td>
<td>66</td>
</tr>
<tr>
<td>11.3.5 Unassigned Orders</td>
<td>67</td>
</tr>
<tr>
<td>12 Classes</td>
<td>68</td>
</tr>
<tr>
<td>12.1 Classes Overview</td>
<td>68</td>
</tr>
<tr>
<td>12.2 Administration Classes</td>
<td>69</td>
</tr>
<tr>
<td>12.3 ClockIn/ClockOut and Logon Classes</td>
<td>77</td>
</tr>
<tr>
<td>12.4 Order Classes</td>
<td>78</td>
</tr>
<tr>
<td>12.5 Dispatch/Cashier Classes</td>
<td>81</td>
</tr>
<tr>
<td>12.6 Management Classes</td>
<td>88</td>
</tr>
<tr>
<td>12.7 Other Classes</td>
<td>96</td>
</tr>
<tr>
<td>13 Database</td>
<td>99</td>
</tr>
<tr>
<td>14 Market Analysis</td>
<td>103</td>
</tr>
<tr>
<td>14.1 Goals of Application</td>
<td>103</td>
</tr>
<tr>
<td>14.2 Reference Analysis</td>
<td>104</td>
</tr>
<tr>
<td>14.3 Market Analysis of Applications</td>
<td>106</td>
</tr>
<tr>
<td>14.4 Market Analysis Other Applications</td>
<td>109</td>
</tr>
<tr>
<td>15 Conclusion</td>
<td>111</td>
</tr>
<tr>
<td>A index.jsp</td>
<td>112</td>
</tr>
</tbody>
</table>
ABSTRACT

THE DESIGN AND IMPLEMENTATION OF A SMALL TO MEDIUM RESTAURANT BUSINESS WEB APPLICATION

By

Edward Gerhardstein

Master of Science in Computer Science

This Project discusses construction implementation of a web application for a small to medium business. This Project examines the Open Source server technologies that support it; along with the software development technologies used to build it. This Project also discusses the J2EE Platform and the Struts framework technologies that are tools used to enhance the development of the web application. The Open Source server technologies used in the Project are Ubuntu Server Operating System, Apache Tomcat Application Server and MySQL Database Server. Before a web application can be developed it needs an Application Server to run on and a Database Server to store its data. An Application Server and a Database Server also need to be installed on a compatible Operating System. The Ubuntu Server, Apache Tomcat Application Server, and MySQL Database Server fulfill the needs of the web application. The Project is built on a J2EE Platform which is a Java based Platform. It provides enhancement to HTML through JSP pages and Java Servlets. It also supports and utilizes JavaScript, XML, and CSS. The Project utilizes the Struts Frameworks. Struts framework enhances the J2EE Platform by providing a common map of Java Class and JSP pages and provides custom tags that can be utilize by the JSP pages.

This paper will discuss the design and development of the application’s 6 functionality modules: ClockIn/ClockOut, Logon, Administrator, Manager, Order, and Casher/Dispatch. The goal of the project was to create a web application for a user to logon and create orders for delivery, carry-out, or dine-in customers. Along with creating orders, the web application needed to able to dispatch orders for delivery and process payments for delivery, carry-out, and dine-in orders. Another goal was to create an administration aspect of the application to maintain items, users, products, etc. The final goal was to give the web application a managerial component to manage employees, inventory, and reports.
Chapter 1
Overview of Pizza Application

When starting this project, the object was to create a web-based application for a small to medium size pizza business. A pizza application was chosen because it was a familiar business, but it could have easily been any type of restaurant or retail business. The goal was to create a web application for a user to logon and create orders for delivery, carry-out, or dine-in customers. Along with creating orders, the web application needed to be able to dispatch orders for delivery and process payments for delivery, carry-out, and dine-in orders. Another goal was to create an administration aspect of the application to maintain items, users, products, etc. The final goal was to give the web application a managerial component to manage employees, inventory, and reports.

A web application gives the user a broader and more dynamic, useful product over other types of server based applications. With a web application there is no need to install a client program on every computer that uses the application. All the web application needs is a web browser and a network or internet connection to connect to the application server. The users logon to the web application with a username and password and have access to certain parts of the web application based on the users’ assigned privileges. This gives some security to the managers and administrators by allowing them to give access to certain parts of the application to some users and not to others.

A web application needs three things, a web application server, a repository or database, and an operation operating systems. The web application server is used to control and run the web application. The repository or database is used to store data collected from the web application. The web application and database servers are software packages that need to be installed on a computer that has an operation operating system preinstalled; so an operation operating system is require as a prerequisite. An operating system also controls all processes running on the server.

The web application, called Pizza Application, was designed and implemented with all inonly open source licenses. It uses Ubuntu as the operating system installed on a Dell server. Apache Tomcat is used as the web application server and MySQL is used as the database server. The web application was developed on a Java J2EE platform with the implementation of Apache Struts framework. The web application utilized JavaServer Pages (JSP) and HTML for the presentation purpose.
Chapter 2
Open Source Licenses Servers

2.1 Open Source License Definition

Open source refers to software that is distributed with its source code so that end-user organizations and vendors can modify it for their own purposes. Most open source licenses allow the software to be redistributed without restriction under the same terms of the license [2]. The pizza web application is developed for small to medium businesses. All the software used for developing this application were available as free open source licenses. By using and providing free open source licenses, production costs can be kept down and makes the web application affordable for a small to medium business owner. For example, a Dell server was purchased for this project, for less than $300.00 and an Ubuntu server operating system was installed for free. Ubuntu server has Apache Tomcat and MySQL prepackaged with it. So Apache Tomcat MySQL is also installed for free. If a restaurant company purchased the pizza application for $1,200.00, the total cost for everything would be less than $1,500.00. That is well below the cost of only buying a web application server and a database server from Microsoft or Oracle.

2.2 Ubuntu

Ubuntu’s first release was on October 20, 2004 and new versions of Ubuntu were released every six months. Ubuntu is an entirely open source operating system built around the Linux kernel [1]. Ubuntu is composed of many software packages, of which the vast majority are distributed under a free software license, making an exception only for some proprietary hardware drivers [6]. Ubuntu was chosen because it is a freely downloadable open source operating system. Ubuntu can be installed in a lean form with only the core kernel module installed. The lean install does not install the GUI and can access the functionalities through command line. By not having the overhead of the GUI, Ubuntu becomes efficient.

2.3 Apache Tomcat

Apache Tomcat started off as a servlet reference implementation by James Duncan Davidson, a software architect at Sun Microsystems, in 1999. He later helped make the project open source and played a key role in its donation by Sun to the Apache Software Foundation. The Apache Ant software build automation tool was developed as a side-effect of the creation of Tomcat as an open source project [20].

Apache Tomcat is a free downloadable open source application server provided by the Apache Software Foundation. Some other application servers, like such as Microsoft and Oracle, charge a licensing fee for each application server installed. Since Tomcat is a free open source license application server, it can be installed on multiple servers without additional fees.
Many of the other application servers have Apache Tomcat embedded into them. JBoss, Websphere, Sun One, and Apache Geronimo are a few application servers that have Apache Tomcat embedded into them. The differences between these and Apache Tomcat is that they are a full implementation of Java J2EE platform and Apache Tomcat is not.

This application does not use the full implementation of the J2EE platform because of the overhead cost of running the full implementation. By not implementing the full version of the J2EE platform, it allows the application to avoid implementing the EJB component of the J2EE platform. The EJB component causes performance problems when running on an application server that implements the full version of the J2EE platform. According to Ron Johnson, EJB introduces a lot of complexity, EJB is not portable between application servers, EJB systems often perform poorly and don’t necessarily scale up well and EJB can make simple things more complicated [5]. By not implementing EJB, the web application has less complexity, more portability and has more scalability. Since the web application is being developed for a small to medium business, the application can do without the overhead caused by EJB.

After the Apache Tomcat server has been set up and installed on the Ubuntu server, a web application can be deployed on it. There are several ways to deploy an Apache Tomcat web application.

One way is to copy an already created document root directory and subdirectory into the directory $CATALINA HOME/webapps/. A document root directory is a directory that is created to store all the files for the application. $CATALINA HOME is the home directory where you installed the Apache Tomcat server is installed. For instance, on my laptop, the $CATALINA HOME is C:\apache-tomcat-6.0.29\. So $CATALINA HOME/webapps/ would be C:\apache-tomcat-6.0.29\webapps\. An example layout of a document root directory is below

PPIIntranet
   -JSPfolder1
   -JSPfolder2
   -JSPfolder3
   -CSS
   -JavaScript
   -Images
   -META-INF
   -WEB-INF
      -classes
      -source
      -web.xml

The second way to deploy an application is to create a web application archive file, a WAR file, and copy it into directory $CATALINA HOME/webapps/ . A WAR file is similar to
a .zip file, as a matter of fact, you can create a .zip file and place all the files and folders from the document root directory into it and rename it to a .war file. An Ant build.xml file can also be used to create a .war file for deployment to the Apache Tomcat Server’s SCATALINA HOME/webapps/. Whichever way used to perform the copy, the Apache Tomcat server will automatically expand the .war file into its document root directory form, and execute the web application when it is restarted.

Another way to deploy your web application is to add a `<Context>` entry in the $CATALINA HOME/conf/server.xml configuration file. In this way you can have your document root directory stored in any other directory outside the $CATALINA HOME/webapps/ directory. Once again the Apache Tomcat Server must be restarted.

Every web application on Tomcat Server needs “The Web Application Deployment Descriptor”. This is stored as /WEB-INF/web.xml in the document root directory. This XML file stores all the information about the components that make the web application, which include servlets, initialize parameters, and other information used in the web application.

For this project, the web application was developed in Eclipse and exported to a .war file. Eclipse is an IDE used mostly for developing Java applications. After the .war file was created it was moved over to the Apache Tomcat server.

### 2.4 MySQL

MySQL is a relational database management system [14]. Original development of MySQL by Michael Widenius and David Axmark began in 1994 [7]. MySQL was first released internally on May 23 1995 [24]. Sun Microsystems acquired MySQL AB on February 28, 2008 [13]. Oracle acquired Sun Microsystems on January 27, 2010. The program runs as a server providing multi-user access to a number of databases.

MySQL is a free, downloadable open source application server provided by the MySQL website. Some other database servers like Microsoft and Oracle charge a licensing fee for each database server installed. Since MySQL is a free open source license database server, it can be installed on multiple servers without additional fees. According to the MySQL website it is ideal for small to medium size businesses. The Standard version was chosen because there are features in the Max version that haven’t been tested thoroughly, and the Debug version has extra debugging code that may cause a reduction in performance.

A database is important to this project because companies store large amounts data. If the data is stored in a flat file system the data would become too cumbersome to keep track of. A database provides a fast and easy way to access data. Since the web application is being developed is for use in a real world application it needs to use a database.

A database is a collection of data stored and organized in an efficient way. There are different kinds of databases, for instance: Flat File Databases, Relational Databases, Distributed Databases, etc.
The idea of a relational database is one that links two-dimensional tables that consist of rows and columns. A user can develop SQL queries to draw relationships between tables to produce an output into a table form. The user does not need to understand the relationship between the tables to generate the output.
Chapter 3
Selected Concepts and Terminologies

3.1 Model-View-Controller (MVC)

Figure 3.1: Figure of Model-View-Controller diagram.

Today in the technical world you hear the acronym MVC come up in books, magazines, and interviews, Model-View-Controller (MVC) is a software architecture or an architectural pattern, to be more precise. It is used currently as part of software engineering theory. It separates the application logic from the user interface. This allows development, testing and maintenance of the logic and user interface independently from each other [23].

The model manages the behavior and data of the application. The model responds to requests for information and responds to instructions for changes that need to be made. In event-driven systems, the model notifies all observers when the information changes have been made [23].

The view renders the model into a form suitable for a user interaction; typically this is embodied in a user interface. There can be multiple views for different purposes in a model. A usual graphical user interface typically has a one to one correspondence with a screen surface that it knows how to render to [23].

The controller receives input from the user and initiates a response by making calls to the model objects. A controller accepts input from the user and instructs the model and graphical user interface to perform actions based on that input [23].

In Java, the Model is the group of Java Classes developed for the application used to store the data. In Java, the View is represented by a Java Server Page. The data between the Classes can be transferred to the page through the HttpServletRequest or HttpSession. In Java, the Controller is an HttpServlet. A servlet controls the flow from the front end to the back end, and transfers that information to another servlet or to the page through the HttpServletRequest or HttpSession [23].
3.2 **JavaScript**

JavaScript is a scripting language that was designed to add interactivity to HTML pages. A scripting language is a lightweight programming language. JavaScript is usually embedded directly into HTML pages. JavaScript is an interpreted language which means that scripts execute without preliminary compilation. Everyone can use JavaScript without purchasing a license [16].

You can embed JavaScript into an HTML page or put it in a separate JavaScript file (xx.js). The advantage of a separate JavaScript file is reusability. All HTML pages and JSP pages have access to the JavaScript functions when they import the JavaScript file into the page.

3.3 **Ajax**

Ajax is short for asynchronous JavaScript and XML. It is not a single language, but a group of technologies used to create an interactive web application. It uses JavaScript to connect the client-side front-end web page to the server-side back-end XML document or Java Servlet. It allows you to dynamically interact with information being displayed, and pull in information from the back-end without reloading the page. It uses the XMLHttpRequest object for asynchronous communication.

3.4 **XML**

XML stands for Extensible Markup Language. XML is a markup language much like HTML. XML was designed to carry data, not to display data. XML tags are not predefined. You must define your own tags. XML is designed to be self-descriptive. You can define XML tags two ways, one way with Document Type Definition (DTD), and the second with an XML Schema [18].

3.5 **DTD**

A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes. A DTD can be declared inline inside an XML document, or as an external reference [15].

3.6 **XML Schema**

XML Schema is an XML-based alternative to DTD. An XML schema describes the structure of an XML document. The XML Schema language is also referred to as an XML Schema Definition (XSD). XML Schema defines elements and attributes that can appear in a document. XML Schema defines which elements are child elements. XML Schema also defines the order of, and number of child elements. XML Schema defines whether an element is empty or can include text. XML Schema defines data types for elements and attributes and XML Schema defines default and fixed values for elements and attributes [17].
3.7 CSS

CSS stands for Cascading Style Sheets and is a language that works with HTML pages to define how to present the content. A Sheet Style defines how to display HTML elements. The CSS specified styles are placed directly into HTML elements, the head of the HTML page, or a separate style sheets [19].

A style sheet contains a number of CSS rules that selects and defines how the elements will be styled in an HTML page [19].

A single file can be used to control the appearance of multiple HTML pages making it easy to maintain. The CSS file can reduce the file size of HTML pages by removing the presentation aspect of the HTML page and placing it into the CSS file. You also have the ability to create style sheets for different media types without the content of the page. External Style Sheets are stored in CSS files (xx.css) [19].
Chapter 4

J2EE Concepts

4.1 J2EE Overview

J2EE is a platform that was designed to develop enterprise applications using Java technology. It also provides standards for developing enterprise applications. J2EE provides several APIs and technologies integrated into its platform.

4.2 JavaBean

A Java Bean is a reusable component model, written in the Java programming language. It also provides portability across platforms. In other words, a Java class that was written to be used on a Sun One Application Server, can also be used on a Websphere Application Server.

4.3 Enterprise JavaBeans (EJB)

An Enterprise Java Bean is similar to a Java Bean, but is used on the server side to encapsulate the business logic of an application. The EJB specification was originally developed in 1997 by IBM and later adopted by Sun Microsystems (EJB 1.0 and 1.1) in 1999 [4]. The EJB provides a standard way to implement business code found in enterprise applications in the back-end [4]. The Enterprise Java Bean was setup to handle, in a standard way, the persistence, transactional integrity, and security aspect of the application. It removes these aspects, leaving the programmers free to concentrate on developing other aspects of the application [4].

4.4 Other J2EE APIs and Technologies

Java Database Connectivity (JDBC) API: JDBC is a set of interfaces which allows Java applications access to any database [3].

Remote Method Invocation (RMI): RMI is an API which allows Java objects to communicate remotely with other objects. This API has the same purpose of as CORBA from OMG [3].

Java IDL: IDL is a standard platform-independent declarative language that is used to define interfaces that object implementations provide and client objects call. Java IDL allows any Java object to communicate with other objects in any language by means of IDL [43].

Java Message Service (JMS): JMS API is a messaging standard that allows J2EE components to create, send, receive, and read messages [3]. Java Transaction API (JTA): JTA allows J2EE components to perform distributed transactions [3].

JavaMail: JavaMail API allows Java components to send and receive emails in a platform-independent way [3].
Java API for XML Processing (JAXP): Extensive Markup Language (XML) is a data format for interchanging structured documents on the Web. JAXP allows Java applications to parse and transform XML documents [3].

4.5 Servlets

Java Servlets are Java classes that are generic server extensions. They can load dynamically to expand the functionality of a server. Most often a servlet is used in web applications. A Servlet is like any other java application except it runs inside a Java Virtual Machine on the server. A servlet operates within the domain of the server and does not require the support for java in the web browsers. Because a servlet runs on the server, it is safe and portable.

The complete servlet specification was created by Sun Microsystems, with version 1.0 finalized in June 1997. James Gosling first thought of servlets in the early days of Java, but the concept did not become a product until Sun shipped the Java Web Server product. As of March 26, 2010, the current version of the servlet specification is 3.0 [21].

The Servlet technologies are contained in the javax.servlet and javax.servlet.http packages. The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing Servlets and Servlet supported Java classes. All Servlet Java classes must implement the Servlet interface.

Some common interfaces and classes used in a web application are HttpServlet, HttpServletRequest, HttpServletResponse, and HttpSession. The Servlet interface defines methods that all servlets must implement. The interface defines methods that initializes the servlet, runs a request, and then removes it from the server. Here are some of the methods below from the Servlet API. [12]

HttpServlet Class: The HttpServlet Class is an abstract class that is used to create a HTTP Servlet subclass that is suitable for Web applications [8].

HttpServletRequest Interface: The HttpServletRequest Interface is used by HttpServlet to provide HTTP request information [9].

HttpServletResponse Interface: The HttpServletResponse Interface is used by HttpServlet to provide HTTP-specific functionality in sending a response [10].

HttpSession Interface: The HttpSession Interface provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user. The servlet container uses this interface to create a session between an HTTP client and an HTTP server [11].

In an HTML web page the <form> tags have parallel input elements that are mapped to the HttpServlet Class methods: doGet, doPost, doPut, and doDelete. tag the input elements
that are most often called are Post and Get. The Get input element is used when the values in the form are being passed as part of the URL. The Post input element is used when there is data being inserted, updated, or retrieved from a data file or database. The data is considered hidden and not visible to the URL box by the user’s browser.

### 4.6 JavaServer Pages (JSP)

Before JSP, and significant developments in Java in the area of web applications, there was the Common Gateway Interface (CGI). Developing web applications in Common Gateway Interface (CGI) was resource-intensive and did not scale well. With the introduction of Servlet technology, Java developers had an easier way to write web applications that generated dynamic content. The task of writing Servlets however still took some effort and required Java expertise.

When Java Server Pages (JSP) technology was introduced, it was developed to be similar to Hypertext Markup Language (HTML), XML, or other document types. This made it easy for web front end developers who were accustomed to Hypertext Markup Language (HTML) to write JSP web applications. They didn’t need to know Java and Servlet programming to develop pages in JSP. JSP also integrated a scripting language into it for Web application clients. Developers, who were educated in HTML and JavaScript, could quickly learn JSP. When a JSP web page is developed, the Web server will convert it to a Servlet.

JSP syntax is a fluid mix of two basic content forms: scriptlet elements and markup. Markup is typically standard HTML or XML, while scriptlet elements are delimited blocks of Java code which may be intermixed with the markup. When the page is requested the Java code is executed and its output is added, in situ, with the surrounding markup to create the final page. Because Java is a compiled language, not a scripting language, JSP pages must be compiled to Java bytecode classes before they can be executed, but such compilation is needed only when a change to the source JSP file has occurred [22].

#### 4.6.1 Scriptlet

A scriptlet starts with `<%` and ends with `%>`. Scriptlet can be used to execute statements in the page’s language. For example we can execute java code in the scriptlet tag.

```
<% scriptlet %>
```

A scriptlet can be used as a declaration.

```
<%! declaration %>
```

To be able to print java variables to the page we can use expression scriptlet.

```
<%= expression %>
```
There is a special scriptlet called directives that can assigned page variables and attributes

<%@ name att1="v1" ... %>

There are also scriptlets for JSP actions. Since JSP are actually converted to a servlet, actions are like methods in the servlet. Any methods that can be used in a servlet can be called in JSP. Some actions include: JSP file, use JavaBean, set properties, and get properties (<jsp:include>, <jsp:useBean>, <jsp:setProperty>, and <jsp:getProperty>).

With JSP a developer has the ability to create custom tags. Custom tags are user-defined JSP language elements that can be used like JSP tags on a JSP page. When the JSP is transformed into a servlet, the custom tags are converted to an operation on the tag handler. A tag handler is a Java object that is created to handle custom tags during the execution of a JSP page. A tag must implement either the Tag or Body Tag interface.

To use the tag the developer needs to do three things. One, develop a tag handler Java class. Two, place an entry in the tag library descriptor. Finally, create tag library descriptor if it doesn’t exist, and use the <%@ taglib %> to import the tag library descriptor in to your JSP page. A tag library descriptor (TLD) is an XML document that describes a tag library. Any tag library descriptor use in the Web application must be stored in the WEB INF directory and have an entry in the web.xml for the tag library descriptor.
Chapter 5

Apache Struts Framework

5.1 Apache Struts Overview

Apache Struts framework is a free open-source license framework for creating J2EE web applications. The Apache Struts framework is designed around MVC architecture. Apache Struts is based on J2EE standard and technologies such as Java Servlets, JavaBeans, ResourceBundles, and XML. It also implements various Apache Commons packages such as BeanUtils and Chain of Responsibility.

The framework provides a request mapping to resources, a response mapping to forward to other resources, and a tag library that can be utilized on the JSP pages.

5.2 ActionServlet

The ActionServlet is the work horse of the framework; it is the central controller of the web application. The Action Servlet processes the user request; figures out what the user wants to do, pulls in data for the user, and then forwards the user to the proper page. The ActionServlet delegates most of its work to the Request Processor and Action classes. The ActionServlet is also responsible for initializing and cleaning up the frameworks resources. One of the resources initializing is the struts-config.xml

The web application can utilize the ActionServlet by placing a servlet mapping in the web.xml file and set it to load on startup.

5.3 Struts Config

Struts framework is configured through struts-config.xml file. Struts framework uses the struts-config.xml to initialize its own resource. The struts-config.xml is similar to the web application deployment descriptor web.xml. Some of the resources initialized are actions, forms, forwards, message resources, and plug-ins. These resources create links to Java classes, ActionForm, Action, and JSP page.

The <form-bean> tag maps ActionForm class to the application. An ActionForm class is basically a Java class that can store data and can be used in a struts application. All the type parameters in a form-bean mapping must be mapped to a class that extends class ActionForm or a sub-class of ActionForm. The data saved in the ActionForm class can be used and displayed in a JSP and/or used in an Action Class.

The <forward> tag maps a path where a request can be forward to. The forward tag can be mapped to a JSP page or Java class.

The <action> tag maps to the Action class to the application. An Action class is a Java
class that is used to process HTTP request. It is used to gather information from the HTTP request and/or ActionForm.

An Action class is used to process an HTTP request. An Action class is the middle-man between the incoming HTTP request and the corresponding business logic. An Action class has two major execute methods that returns an ActionForward. The majority of the web applications uses the HttpServletRequest execute() method. The other execute() method is provided for applications that do not require the HTTP protocol.

A `<message-resources>` tag maps the message resources for the web application. These resources are usually a .properties file. A resources file can be used to store error messages and messages displayed on a JSP page.

Struts framework also provides custom tags to use in the JSP pages and can be accessed through the struts tag library descriptors. Some examples are Struts-bean.tld, Struts-html.tld, and Struts-logic.tld.
Chapter 6

Pizza Application Overview

6.1 Design Layout

To begin exploring the design project the layout of the project needs to be discussed. The project is a restaurant application, it could be any restaurant application, but for the purpose of this project it is a pizza application. As discussed earlier in the Implementation of Apache Tomcat section there is a root directory created called PPINTRANET. Subfolders for source code and web content are created in the root directory. The web application is then divided into files and folders based on functionality.

In the source folder packages are created for all the functionalities of the web application. For example a package is created for orders and all the classes created for orders are places into it. An action package is created to place all the action classes. It makes more sense now, after the application has been develop, to keep the Struts Actions and Action-Forms classes that are used in the functionality to be in the same place as the other classes.

In the web content folder, other folders are created according to the web page functionality. Different JSP page folders are created for this application and JSP pages placed into them based on the JSP page purpose. There are other folders that are created for images, scripts, themes, and includes. Images that are used for the web application are placed into the image folder, JavaScript files that are placed in the scripts folder, CSS files are placed into the theme folder, JSP pages for the footer and header are placed into the include folder. The footer.jsp and the header.jsp are specialized JSP pages that are used in every JSP created in the application.

Another folder created in the web content folder is WEB-INF folder. The WEB-INF folder is where the web.xml and the structs-config.xml are stored. Struts TLD files are also stored in the WEB-INF folder. There is also a folder called classes in the WEB-INF folder that stores all the compiled Java Classes. Another folder called lib in the WEB-INF folder is used to store all the JAR files used in the application like struts.jar and mysql.jar. By keeping files, classes, and pages in separate folders, it keeps the project organized and easy to follow. It also makes developing the application easier by keeping each functionality that is being developed in their own common centralized folder.

6.2 Workflow

In the project, there are two basic workflows for each page of the application that the pages follow. There are two workflows because at the start of the project the web pages were created one way and over the course of working on the project a new way was learned to create the web pages that accomplishes the same outcome. It’s not that one is better than the other; it’s just two different ways to do the same thing. One way is more Java code integrated into the JSP page. The other way takes advantage of the features in the JSP page
and the Struts Framework, the Java code is removed from the JSP page altogether. The application utilizes many of the features of the Struts Framework and the JSP page, but it doesn’t utilize all the features. Although the other features were interesting, they weren’t necessary to complete the project. Since most of the application was already coded, it would have taken more time to implement the other features on all the JSP pages.

The first workflow created a JSP page and an XXXAction class for that particular page. An action-mapping for the XXXAction and a global-forward for the JSP page is placed into the struts-config file. The action-mapping had the path and type fields set in the action to the XXXAction class. The global-forward had the name and path fields set in the forward to the JSP page. An XXXManagerBean class was created and used to make connections to the database to run SQL queries. In XXXManagerBean class methods are created to run SQL queries such as SELECT, UPDATE, and DELETE, against a table or tables in the particular database schema. The last thing that was created for each page was XXXValidationBean class for validating the variables retrieved from the JSP pages. Although variables entered on the JSP page can be validated within the JSP page, certain variables have to be validated against the database. The XXXValidationBean class gives the application a way to validate all the variables from the JSP page and returns an error code if a variable is not valid.

In the XXXAction there is an action variable that is correlated to a button or a hyperlink on the JSP page. For each button or hyperlink the user clicks on, there is a correlated operation that is performed and the user is forwarded to a JSP page base on value of action variable. This makes it easy to perform tasks and move from JSP page to JSP page. This also keeps the logic behind the web application hidden form the end user. Figure 6.1 is a diagram of the first workflow architecture.

The other workflow is a little more elaborate. Just like above the application has a JSP page,
an XXXAction class and an XXXManagerBean class created for that particular page, but an XXXActionForm and an XXXBean class is also created. The XXXForm class is used to populate and retrieve data fields on the JSP pages. The XXXBean class is used to store array values to be used on the JSP page. An action-mapping for the XXXAction class is placed in the struts-config file, with the fields set for the path, type, name and scope for the action. The action mapping may also have a forward map to the JSP page with the name and path fields set within the action. If the forward is not mapped in the action, the JSP page is mapped in the global-forward. The XXXActionForm used in this workflow extends the ValidatorForm. The ValidatorForm is a Struts ActionForm that provides basic field validation. It also has a method called validate() that can be used to validate attributes that were set by the JSP page. If the values are not valid, then the web application is returned to the JSP page with an error message.

In this workflow the XXXActionForm contains an action variable that is correlated to the buttons or the hyperlinks on the JSP page. For each button or hyperlink the user clicks on, there is a correlated operation that is performed and the user is forwarded to a JSP page based on value of action variable. The XXXAction class stills performs operations base on the action value and forwards the user to the next JSP page. This makes the logic behind the web application even more hidden from the end user. In this case the XXXActionForm is integrated into the JSP. This means that the JSP doesn’t have to pull a Java class into the page to populate its fields, the JSP has direct access to all the variables of the XXXActionForm that have getter and setter methods. Figure 6.2 is a diagram of the second workflow architecture.

![Diagram of the second architecture.](image)

6.3 JSP Page formats - Index.jsp/Templates

There are several ways the application can format JSP pages, one way is a template JSP page. An index.jsp is used as a template in the application. A template is a page that
is used to create a format or layout for all the pages to follow. There can be more than one template in the application depending on the need of each section of the application. Some of the common layouts used are: header, side, body, and footer; or header, body, and footer. The layout this application uses is header, body, and footer. The index.jsp used the `<jsp:include>` to bring in the header, body, and footer JSPs. The header and footer are static JSP pages, while the body was dynamic, depending on what part of the web-application was in use.

The index.jsp is also used to pull in the common elements used by all JSP pages. The index.jsp pulls in all the JavaScript files and CSS files used throughout the application. It also set the metadata elements for all the JSP files. Appendix A is sample code of the index.jsp used in the application.

Another way some JSP pages are formatted throughout the application is with a hyperlink page. The hyperlink page has a list of links to other JSP pages. The hyperlinks are not mapped directly to the link JSP page, but are forwarded through a common NavAction class. The application uses a common NavAction class to insure security to the functionalities. The users cannot access the JSP pages associated to functionality unless they have the right access user role. This is done throughout the application and all linked pages have a common NavAction class to forward the users from one JSP page to another. The NavAction class checks for the appropriate role access from the user that is loaded into the memory, then forward onto the next JSP page.

The final way a JSP page is formatted throughout the application is more like a form format. When working with these JSP pages the user either enters values or selects values in the form and then clicks on a submit button. The application forwarded to an XXXAction class and the XXXAction class calls a XXXManagerBean class to make connection to the database and manipulate data in the database. The XXXAction then forwards the user to the next JSP page.

### 6.4 JSP Page Divisions

The application is divided into six parts, or functionalities. ClockIn/Clockout, Logon, Administrator, Manager, Order, and Cashier/Dispatch. The Clockin/Clockout and Logon functionalities give the employees and users a simple starting access point to the application. The other four give the user a little more functionality based on user role permissions. The application is divided into functionalities for simplicity, to list all tasks and functions on one page would be too confusing to the user. By dividing the application into functionalities, it makes it easier for the user to maneuver through the application and perform the task the user wants to do. Figure 6.3 is a diagram of the application division.

There is a difference between a user and an employee. An employee clocks in and out of the application. A user can logon to the web application. A user can be associated to an employee but doesn’t have to be. For example Jane Doe is an employee and can clock in with employeeid “janedoe”. Jane Doe can logon to the application using userid “janed”.
There are employees that only need to clock in and out, but do not need to logon the web application. For instance, the cooks just work in the kitchen; therefore, they only need to clock in and out. The ClockIn/ClockOut functionality was developed for that purpose. The users can have certain access privileges based on their roles. For instance, a user that has a manager role can access the Management functionality of the application.

Individual users are created with roles assigned to them instead of creating one user logon per role to track changes made to the application. Each table in the database has columns for created time, created date, created by user, and created by page. Each table in the database also has columns for last maintain time, last maintain date, last maintain by user, and last maintain by page. These columns are used to track changes made to the application, it keeps track of who is making the change, what date the changes were made, what time the changes were made and what part of the application the changes were made. This also is useful in debugging the application during developing and maintenance of the application. It can be used to see if values in the database are being set wrong, and can tell the developer what part of the application is setting the values wrong. The developer can go to that particular part of the application and make the corrections accordingly.

When entering the application, the user is forwarded to the nav.jsp page. The nav.jsp page is the access point to the application. The nav.jsp page has links for clock and logon which forwards the user to the clockDisplay.jsp page and the logon.jsp page respectively. When first designing the application a question came up about whether a user should logon on to the applications first and then clock in and or should the user clock in first and then logon onto the application?. An employee needs to clock in and out of work, but doesn’t necessarily need to logon to the application. The user should clock in before they can logon to the application. The nav.jsp page was design with the links clock and logon to give the user a way to do both. Figure 6.4 is a screen shot of the nav.jsp page.
Figure 6.4: Screenshot of the nav.jsp page.
Chapter 7

ClockIn/Clockout and Logon Functionality

7.1 ClockIn/Clockout Functionality

When the employee clicks on the clock link, the employee is forwarded to the clockDisplay.jsp. The clockDisplay.jsp is the access point to the ClockIn/Clockout functionality of the application. ClockIn/Clockout functionality is basically a way for employees to clock in and out. It takes two variables Username and Password and when the user clicks on the Clock-In/Out the user proceeds to clock in and out of the application. If the employee is not clocked in, the application will clock the employee in. If the employee is already clocked in, the application will clock the employee out. After the employee is clocked in or out, the employee is forwarded back to the nav.jsp. A reset button was added to clear the variables Username and Password, and a “Back to Main Menu” was added to return the user back to nav.jsp page. Figure 7.1 is a screenshot of clockDisplay.jsp page.

7.2 Logon Functionality

When the user clicks on the logn link the user is forwarded to the logon.jsp. The logon.jsp is the access point to the Logon functionality of the application. The Logon functionality is a way for a user to access the application. The design is a typical and simple Logon page design, with a Username and Password fields. The user enters the Username and Password and click Logon button to log into the application. When the user clicks on the Logon button, first the application checks to see if a user exists with the userid and password. If the user exists, the application loads all the user’s information and privileges into a UserBean object and stores the object into the memory, if the user does not exist, the user is forwarded back to the logon.jsp page. A reset button was added to clear the
variables Username and Password, and a “Back to Main Menu” was added to allow the user to return back to logon.jsp page. Figure 7.2 is a screenshot of logon.jsp page.

![Screenshot of logon.jsp page](image)

**Figure 7.2: Screenshot of the logon.jsp page.**

After the user is validated, the user is forwarded to the Main Menu JSP page. The Main Menu JSP page lists the functionalities of the application. The functionality list was designed to make it easy navigate to through the application by grouping together tasks that have common functions. For example Order functionality groups together all tasks that deal with order taking. The main.jsp page was designed only to display a list of functionalities that the user’s role has access to. The list is based on the user’s role and only shows the functionality that the user’s role has access to. The list includes Orders, Dispatch/Cashier, Management, and Administration. The list is hyperlinked to the functionalities main JSP page. The Orders hyperlink is linked to the Order functionality main page orderMain.jsp page, the Dispatch/Cashier hyperlink is linked to the Dispatch/Cashier functionality main page dispCashMain.jsp page, the Management hyperlink is linked to the Management functionality main page manageMain.jsp page, and the Administration hyperlink is linked to the Administration functionality main page aminMain.jsp page. Figure 7.3 is a screenshot of is a screen shot of the main.jsp page.
<table>
<thead>
<tr>
<th>Pizza Restaurant</th>
<th><strong>Main Menu</strong></th>
<th>Pizza Application</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Orders</td>
<td>Work with Orders</td>
<td></td>
</tr>
<tr>
<td>- Dispatch/Cashier</td>
<td>Dispatch or Cash Out Orders</td>
<td></td>
</tr>
<tr>
<td>- Management</td>
<td>Management Functions</td>
<td></td>
</tr>
<tr>
<td>- Administration</td>
<td>Product and User Administration</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.3: Screenshot of the main.jsp page.
8.1 Administrator Functionality Description

Administrator Functionality gives the users a way to administer the application. Administrator Functionality gives the user a way to add, update, and remove users, employees, products, and other items to the web application. In order for the application to be useful for the small to medium business, these actions are required. Administrator Functionality gives the users a broad way of doing these tasks. In order to view any of the Administrator Functionalities, the user must have the admin user role.

When the user clicks on the Administrator link from the main.jsp page, the user is forwarded to the adminMain.jsp page. The adminMain.jsp page consists of a list of administrator hyperlinks. The Administrator Functionality module of the application is quite large, and there are many tasks that need to be done to maintain the application. To make it easier for the user to navigate through, the Administrator Functionality of the application was divided into four sections. The tasks are divided into these four sections based on which section the tasks are mostly associated with. For Example, create a new product or category is associated to the Product section. The list contains links to Product, Inventory, Employee, User and Main. Product Administration provides a way to administrate items associated with product objects in the application. The Product hyperlink is linked to the Product Administration entry page, adminProduct.jsp. Inventory Administration provides a way to administrate items associated with inventory objects in the application. The Inventory hyperlink is linked to the Inventory Administration entry page adminInventory.jsp. The Employee hyperlink is linked to the Employee Maintenance entry page, employeeMaintenance.jsp. The User hyperlink is linked to the User Maintenance entry page, userMaintenance.jsp. The Main link forwards the user back to the application main.jsp. Figure 8.1 is a screenshot of the adminMain.jsp page.

8.2 Product Administration

When the user clicks the Product link from the adminMain.jsp page, the user is forwarded to the adminProd.jsp page. A product is an object that is being sold in the application. Product Administration gives a user a way to maintain these objects. Product Administration is not just administering product objects in the application, but also other objects that are associated to the product objects. The Product Administrator page gives the user a way to maintain aspects of the application that are associated to a product object. The adminProd.jsp page consists of a list of product administrator hyperlinks. The adminProd.jsp was designed to maintain items that are associated with the product objects. The list gives the user a direct link to a group of objects that is being maintained. The whole idea behind this link page and the previous link page is to make it easier for the user to navigate through the application to get to the group of items that they want to maintain. With three clicks on three different hyperlinks the user is in the group of objects the user
This makes it easier for the user because the user doesn’t have a long list of tasks to scroll through and the user doesn’t have to make more than three or four clicks to get to the task they want to perform. The list contains links to Size, Category, Product, Product Crust, Product Items, Administration Main, and Main. The Size hyperlink is linked to the Size Maintenance entry page, sizeMaintenance.jsp. The Category hyperlink is linked to the Category Maintenance entry page, categoryMaintenance.jsp. The Product Crust hyperlink is linked to the Crust Maintenance entry page, crustMaintenance.jsp. The Product Items hyperlink is linked to the Product Item Maintenance entry page, itemMaintenance.jsp. The Administration Main and Main links forwards the user back to the application adminMain.jsp and main.jsp pages respectively. Figure 8.2 is a screenshot of the adminProduct.jsp page.

Figure 8.2: Screenshot of the adminProduct.jsp page
8.2.1 Size Maintenance

When the user clicks on the Size link from the adminProduct.jsp page, the user is forwarded to the sizeMaintenance.jsp page. The Size Maintenance page gives the user a way to maintain product size types in the application. The sizeMaintenance.jsp page was designed to make it simple for the users to choose what task they want to do in Size Maintenance. The sizeMaintenance.jsp page displays a list of sizes in the application, this gives the user a way to view the size currently in the application to make the decision to add a new size, delete a size, or edit a size item. The sizeMaintenance.jsp page has buttons for Add New, Change, Delete and Product Admin Menu placed across the top of list. Figure 8.3 is a screenshot of the sizeMaintenance.jsp page.

![Size Maintenance Screenshot](image)

When the user clicks on the Delete button, the select size option is removed from the application. When the user clicks on Product Admin Menu button the user is redirected back to the adminProd.jsp page. The user can create a new size by clicking on the Add New button. When the user clicks on the Add New button the user is forwarded to the sizeAdmin.jsp page. The sizeAdmin.jsp page gives the user a way to add a new size to the application. When the user clicks on the Change button the user is forwarded to sizeAdmin.jsp page with the selected size. The sizeAdmin.jsp page gives the user a way to make changes to the selected size in the application.

On sizeAdmin.jsp there are several variable fields to change or add new size item. The Size ID field is used to create a unique identifier for a size item. When adding new size, this field is used to verify that no other size exists with the same identifier. The Size ID is also used to associate the size item to other objects in the application. For example all product items are associated to a size item. The Size Description gives the user a more detailed description of the size item and is used when an item is being displayed in other parts of the application. The Size Active option field can set the size item to be activated.
and deactivated in the application. When the size item is deactivated, the size item does not display in other parts of the application.

The Add New and Change hyperlink links on the sizeMaintenance.jsp page are both forwarded to sizeAdmin.jsp. For Add New size, the sizeAdmin.jsp displays a Next and a Cancel button across the top. For Change size, the sizeAdmin.jsp displays an Update and a Cancel button across the top sizeAdmin.jsp and the Employee Id field is visible but not editable. When the user clicks on the Next button, a new size is added to the application. When the user clicks on the Update button, the changes made to the size item is updated in the application. When the user clicks on the Cancel button the user is redirect back to the sizeMaintenance.jsp page. Figure 8.4 are a screenshots of the sizeAdmin.jsp page.

![Figure 8.4: Screenshots of the sizeAdmin.jsp page](image)

### 8.2.2 Category Maintenance

When the user clicks on the Category link from the adminProduct.jsp page, the user is forwarded to the categoryMaintenance.jsp page. The Category Maintenance page gives the user a way to maintain product category types in the application. The categoryMaintenance.jsp page displays a list of categories in the application so the user can view the current categories that exist in the application. The categoryMaintenance.jsp page has Work With, and Product Admin Menu buttons placed across the top of the page. The Product Admin Menu button forwards the users back to the adminProd.jsp page. The categoryMaintenance.jsp page doesn’t have an Add New or Delete button because a category is linked to a type of flow the application traversing through when adding items to an order. Depending on which category item the order item is associated to, the order could traverse through one, two, or three JSP pages to add an order item to the order. Also, every product in the application is associated with a category item, and a decision was made to make the category item the top level item and a static value. Since it is a static value, the user can’t remove it from the application; therefore a Delete button was not added to the categoryMaintenance.jsp page. If a category item was ever removed, the database integrity would be jeopardized because both product items and order items are dependent on category. Figure 8.5 below is a screenshot of the categoryMaintenance.jsp page.

![Figure 8.5: Screenshot of the categoryMaintenance.jsp page](image)
The Work With button on the categoryMaintenance.jsp forwards the user, with the selected category, to the categoryAdmin.jsp. The categoryAdmin.jsp displays the Category ID, Category Description, and Category Short Name variables fields and the Category Active selected options. The Category ID field is used to identify the category and other objects in the application associated to it. Since other objects are associated to the Category ID field, the user can not edit the Category ID field. The Category Description variable field is used to describe the category item and is used for display purpose throughout the application. The Category Short Name field is a short description of the category item and is used primarily to display the category in the order summary and printing purpose. The Category Active option field can set the category item to be active (yes) or inactive (no). By setting the category item as inactive, the category is not used in the order taking process. When the user clicks on the Cancel button, the user is forwarded back to categoryMaintenance.jsp. When the user clicks on the Update button, the changes made to the category will be updated in the application. Figure 8.6 is a screenshot of the categoryAdmin.jsp page.

### 8.2.3 Product Maintenance

When the user clicks on the Product link from the adminProduct.jsp page, the user is forwarded to the productMaintenance.jsp page. The Product Maintenance page gives the user a way to maintain the products in the application. The productMaintenance.jsp page displays a list of products in the application. The list was designed to give the user a view of all the products that are available in the application and select a product to change or delete. The productMaintenance.jsp page has Add New, Change, Delete and Product Admin Menu buttons placed across the top of list. The Product Admin Menu button redirects the users back to the adminProd.jsp page. The Delete button gives the user a way to removes a selected product from the application. The Add New button gives the user a way to add a new product to the application. The Change button gives the user a way to make changes to the selected product. Figure 8.7 is a screenshot of the productMaintenance.jsp page.
When the user clicks on the Add New and Change button, on the productMaintenance.jsp page the user is forwarded to the productAdmin.jsp. The Change button forwards the user, with the select product from the list, to productAdmin.jsp page. The productAdmin.jsp page gives the user a way to add a new product or make changes to the select product in the application. The productAdmin.jsp displays variable fields Product ID, Product Description, Product Size, Product Price, and Item Category. There is also a Product Active option field to set the product as active or inactive. The Product ID variable field is a unique identifier and is used to associate the product item to other parts of the application. If the Product ID value already exists in the application, the user is forward back to productAdmin.jsp page with an error message. The Product Description field is used for display purposes throughout the application and gives a description of the product. The Product Size field gives the user a way to assign a size to the product. The Product Price field allows the user to...
to enter a price for that particular product. The Product Active option allows a user to set the product as active or inactive. This is useful because if there are order items that are associated with the product, deleting the product would jeopardize the database integrity. By allowing the user to set the product as inactive, the product will not be displayed throughout the application, but will not be removed from the database. For the Add New product, the productAdmin.jsp displays a Next and a Cancel button across the top. For the Change product, the productAdmin.jsp displays an Update and a Cancel button across the top of the productAdmin.jsp page and the Product ID field is visible, but is not editable. When the user clicks on the Next button, a new product is added to the application. When the user clicks on the Update button, the changes to the selected product is updated to the application. When the user clicks on the Cancel button, the user is forwarded back to the productMaintenance.jsp. Figure 8.8 depicts screenshots of the productAdmin.jsp page.

![Figure 8.8: Screenshots of the productAdmin.jsp page](image)

### 8.2.4 Crust Maintenance

When the user clicks on the Product Crust link from the adminProduct.jsp page, the user is forwarded to the crustMaintenance.jsp page. The Crust Maintenance page gives the user a way to maintain the product crust in the application. The crustMaintenance.jsp page displays a list of products in the application. This allows the user to view a list of crust items available to the application and select a crust item to delete or make changes to. The crustMaintenance.jsp page has Add New, Change, Delete and Product Admin Menu buttons placed across the top of list. The Product Admin Menu button forwards the user back to the adminProd.jsp page. The Delete button gives the user a way to remove the selected crust item from the application. The Add New allows the user to add a new crust item to the application. The Change button gives the user a way to make changes to the selected crust item. Figure 8.9 is a screenshot of the crustMaintenance.jsp page.

When the user clicks on the Add New and Change button on the crustMaintenance.jsp, the user is forwarded to the crustAdmin.jsp. When the user clicks on the Change button, the user is forwarded with the selected product crust from the list on crustMaintenance.jsp page. The crustAdmin.jsp page gives the user a way to add a new product crust or make...
Figure 8.9: Screenshot of the crustMaintenance.jsp page.

8.2.5 Item Maintenance

When the user clicks on the Product Items link from the adminProduct.jsp page, the user is redirected to the itemMaintenance.jsp page. The Item Maintenance page gives the user a way to maintain the product items in the application. A product item is an optional item that can be added to the product during the order process. For instance a cheese pizza product can add a pepperoni item as a topping. The itemMaintenance.jsp page displays a list of product items in the application. The list allows the user to view a list product items in the application and allows the user to select a product item to edit or delete. The itemMaintenance.jsp page has Add New, Change, Delete and Product Admin Menu buttons.
Figure 8.10: Screenshots of the crustAdmin.jsp page

placed across the top of list. The Product Admin Menu button redirects the user back to the adminProd.jsp page. The Add New button gives the user the ability to add a new product item to the application. The Change button gives the user the ability to make changes to the selected product item in the application. The Delete button removes the selected product item from the application. Figure 8.11 is a screenshot of the itemMaintenance.jsp page.

Figure 8.11: Screenshot of the itemMaintenance.jsp page.

The Add New and Change button on the itemMaintenance.jsp forwards the user to the itemAdmin.jsp. The Change button forwards the user, with the selected item from the list, to itemAdmin.jsp page. The itemAdmin.jsp page gives the user a way to add a new product item or make changes to the selected product crust in the application. The itemAdmin.jsp displays variable fields Item ID, Item Description, Item Size, Item Price, and Item Cate-
8.3 Inventory Administration

When the user clicks the Inventory link on the adminMain.jsp page, the user is forwarded to the adminInventory.jsp. Inventory is supplies that are used to produce the product sold. The Inventory can be associated to the products, the product items, and crust items. Inventory Administration gives the user the ability to maintain aspects of the application that are associated to an inventory object in the application. The adminInventory.jsp
page consists of a list of inventory administrator hyperlinks. The adminInventory.jsp was designed to maintain items that are associated to the inventory objects. This list gives the user a direct link to a group of items that is being maintained. The list contains links to Inventory Entry, Inventory Product, Inventory Items, Inventory Crust, Admin Main, and Main. The Inventory Entry hyperlink is linked to the Inventory Maintenance entry page, inventMaintenance.jsp. The Inventory Product hyperlink is linked to the Inventory Product Maintenance entry page, inventProdMaintenance.jsp. The Inventory Items hyperlink is linked to the Inventory Item Maintenance entry page, inventItemMaintenance.jsp. The Inventory Crust hyperlink is linked to the Inventory Crust Maintenance entry page, inventCrustMaintenance.jsp. The Administration Main and Main links redirects the user back to the application adminMain.jsp and main.jsp pages respectively. Figure 8.13 is a screenshot of the adminInventory.jsp page.

<table>
<thead>
<tr>
<th>Pizza Restaurant</th>
<th>Inventory Admin Main</th>
<th>Pizza Application</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Admin Inventory Main:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▶ Inventory Entry</td>
<td>Add and Remove Inventory Items</td>
<td></td>
</tr>
<tr>
<td>▶ Inventory Product</td>
<td>Associate Inventory Items to Products</td>
<td></td>
</tr>
<tr>
<td>▶ Inventory Items</td>
<td>Associate Inventory Items to Product Items</td>
<td></td>
</tr>
<tr>
<td>▶ Inventory Crust</td>
<td>Associate Inventory Items to Product Crust</td>
<td></td>
</tr>
<tr>
<td>▶ Admin Main</td>
<td>Back to Admin Main</td>
<td></td>
</tr>
<tr>
<td>▶ Main</td>
<td>Back to Main</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.13: Screenshot of the adminInventory.jsp page.

### 8.3.1 Inventory Maintenance

When the user clicks on the Inventory Entry link from the adminInventory.jsp page, the user is forwarded to the inventoryMaintenance.jsp page. The Inventory Maintenance page gives the user a way to maintain the inventory objects in the application. The inventoryMaintenance.jsp page displays a list of inventory objects in the application. The list provides the user the ability to view all the inventory items available to the application. It also allows the user to select an inventory object that the user would like delete or change. The inventoryMaintenance.jsp page has Add New, Change, Delete and Inventory Menu buttons placed across the top of list. The Inventory Menu button forwards the user back to the adminInventory.jsp page. The Add New button gives the user the ability to add a new inventory object to the application. The Change button gives the user the ability to make changes to the selected inventory object in the application. The Delete button removes the selected inventory object from the application. Figure 8.14 is a screenshot of the inventoryMaintenance.jsp page.
The Add New and Change buttons on the inventoryMaintenance.jsp forwards the user to the inventoryAdmin.jsp. The Change button forwards the user with the selected inventory object from the list to inventoryAdmin.jsp page. The inventoryAdmin.jsp page gives the user a way to add a new inventory object or make changes to the selected inventory object in the application. The inventoryAdmin.jsp displays variable fields: Inventory ID, Inventory Description, and Item Message. There are also Inventory Active option fields to set the inventory as active or inactive. The Inventory ID field is the unique identifier of the inventory object and is a way to associate the inventory object to other items in the application. When adding a new inventory object, if an inventory object already exists with the same Inventory ID, the user is redirected back to the inventoryAdmin.jsp page with an error message. The Inventory Description field is a description of the inventory object and is for display purpose throughout the application. The Inventory Message field is a message to inform a user that the inventory object supply is getting low. For the Add New item, the inventoryAdmin.jsp displays a Next and a Cancel button across the top. For the Change item, the inventoryAdmin.jsp displays an Update and a Cancel button across the top of the inventoryAdmin.jsp page and the inventory ID field is visible but not editable. When the user clicks on the Next button, a new inventory object is added to the application. When the user clicks on the Update button, the changes made to the inventory object is updated to the application. When the user clicks on the Cancel button, the user is forwarded back to the inventoryMaintenance.jsp. Figure 8.15 is screenshots of the inventoryAdmin.jsp page.

8.3.2 Inventory Product Maintenance

When the user clicks on the Inventory Product link from the adminInventory.jsp page, the user is forwarded to the inventProdMaintenance.jsp page. The Inventory Product Maintenance page gives the user away to associate an inventory objects with product objects (inventory/product object) in the application. The inventory/product association is a way to keep track of each inventory object when a product object is sold. When a product is sold, the store inventory on hand is reduced based on the inventory object that is associated with the product object. The inventProdMaintenance.jsp page displays a list of inventory/product object in the application. The list allows the user to view all the inventory/product objects in the application. The list also allows the user to select the inventory/product object that the user would like to change or delete. The inventProdMainte-
The itemAdmin.jsp page has Add New, Change, Delete and Inventory Menu buttons placed across the top of the list. The Inventory Menu button forwards the user back to the adminInventory.jsp page. The Delete button removes the selected inventory/product object from the application. The Add New allows the user to add a new inventory/product object to the application. The Change button allows the user to make changes to the inventory/product object in the application. Figure 8.16 is a screenshot of the inventProdMaintenance.jsp page.

![Image](image1.png)

**Figure 8.15: Screenshots of the itemAdmin.jsp page**

The Add New and Change button on the inventProdMaintenance.jsp forwards the user to the inventoryProduAdmin.jsp. The Change button forwards the user with the select inventory/product object to inventProdAdmin.jsp page. The inventProdAdmin.jsp page gives the user a way to add a new inventory/product object or make changes to the select inventory/product object in the application. The inventProdAdmin.jsp displays variable fields: Inventory ID, Product ID, Size, Inventory Description, and Item Message. There are also Inventory Product Active option fields to set the item active or inactive. The Inventory ID field allows the user to select an inventory object that is going to be associated to a product.
object. The Product ID field is used to select a product object that is going to be associated to the inventory object. The Size field allows the user to select what size is associated with the inventory/product object. The Inventory Product Description field allows the user to provide a description of the inventory/product object for displaying across the application. The Inventory amount allows the user to set the amount the inventory is reduced by when a product is sold. For the Add New item, the inventProdAdmin.jsp displays a Next and a Cancel button across the top. For the Change item, the inventoryAdmin.jsp displays an Update and a Cancel button across the top of the inventoryAdmin.jsp page and the inventory ID field is visible but not editable. When the user clicks on the Next button, a new inventory/product object is added to the application. When the user clicks on the Update button the inventory/product object changes are updates to the application. When the user clicks on the Cancel button, user is forwarded back to the inventProdMaintenance.jsp. Figure 8.17 is screenshots of the inventProdAdmin.jsp page.

![Screenshot of inventProdAdmin.jsp](image)

Figure 8.17: Screenshots of the inventProdAdmin.jsp page

### 8.3.3 Inventory Item Maintenance

When the user clicks on the Inventory Item link from the adminInventory.jsp page, the user is forwarded to the inventProdMaintenance.jsp page. The Inventory Item Maintenance page gives the user a way to associate inventory objects with product item objects (inventory/product item object) in the application. The inventory/product item association is a way to keep track of each inventory object when a product item object is added to an order. When a product item is added to an order, the store inventory on hand is reduced based on the inventory object that is associated with the product item object. The inventProdMaintenance.jsp page displays a list of inventory/product item objects in the application. The List allows the user to view the inventory/product item objects that are in the application. The users can then select an inventory/product item object from the list to make changes to the object or delete the object from the application. The inventProdMaintenance.jsp page has: Add New, Change, Delete and Inventory Menu buttons placed across the top of list. The
Add New button allows the user to add a new inventory/product item object to the application. The Change button allows the user to make changes to the selected inventory/product item object in the application. The Inventory Menu button forwards the user back to the adminInventory.jsp page. The Delete button removes the selected inventory/item object from the application. The figure below is a screen shot of the inventProdMaintenance.jsp page. Figure 8.18 is a screenshot of the inventProdMaintenance.jsp page.

<table>
<thead>
<tr>
<th>Pizza Restaurant</th>
<th>Inventory Product Maintenance</th>
<th>Pizza Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add New</td>
<td>Change</td>
<td>Delete</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inventory Product Item Main:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inventory ID</strong></td>
</tr>
<tr>
<td>Pepperoni</td>
</tr>
</tbody>
</table>

Figure 8.18: Screenshot of the inventProdMaintenance.jsp page.

The Add New and Change button on the inventProdMaintenance.jsp forwards the user to the inventoryProdAdmin.jsp. The Change button forwards the user with the select inventory/product object to inventProdAdmin.jsp page. The inventProdAdmin.jsp page gives the user a way to add a new inventory/product object or make changes to the select inventory/product object in the application. The inventProdAdmin.jsp displays variable fields: Inventory ID, Product ID, Size, Inventory Description, and Item Message. There are also Inventory Product Active option fields to set the item active or inactive. The Inventory ID field allows the user to select an inventory object that is going to be associated to a product object. The Product ID field is used to selected product object that is going to be associated to the inventory object. The Size field allows the user to select what size is associated with the inventory/product object. The Inventory Product Description field allows the user to give a description of the inventory/product object for displaying across the application. The Inventory amount allows the user to set the amount the inventory is reduced by when a product is sold. For the Add New item, the inventProdAdmin.jsp displays a Next and a Cancel button across the top. For the Change item, the inventoryAdmin.jsp displays an Update and a Cancel button across the top of the inventoryAdmin.jsp page and the inventory ID field is visible but not editable. When the user clicks on the Next button, a new inventory/product object is added to the application. When the user clicks on the Update button the inventory/product object changes are updates to the application. When the user clicks on the Cancel button, user is redirected the back to the inventProdMaintenance.jsp. Figure 8.19 is screenshots of the inventProdAdmin.jsp page.

### 8.3.4 Inventory Crust Maintenance

When the user clicks on the Inventory Crust link from the adminInventory.jsp page, the user is forwarded to the inventProdMaintenance.jsp page. The Inventory Crust Maintenance page gives the user a way to associate inventory objects with product crust objects.
Figure 8.19: Screenshots of the inventProdAdmin.jsp page

(inventory/product crust object) in the application. The inventory/product crust association is a way to keep track of each inventory object when a product crust object is added to an order. When a product crust is added to an order, the store inventory on hand is reduced based on the inventory object that is associated with the product crust object. The inventProdMaintenance.jsp page displays a list of inventory/product crust objects in the application. The list allows the user to view the inventory/product crust objects that are in the application. The user can then select an inventory/product crust object from the list to make changes to the object in the application or delete the object from the application. The inventProdMaintenance.jsp page has: Add New, Change, Delete and Inventory Menu buttons placed across the top of list. The Inventory Menu button forwards the user back to the adminInventory.jsp page. The Delete button removes the selected inventory/product crust object from the application. The Add New button allows a user to add a new inventory/product crust object to the application. The Change button allows a user to make changes to the selected inventory/product crust object in the application. Figure 8.20 is a screenshot of the inventProdMaintenance.jsp page.

Figure 8.20: Screenshot of the inventProdMaintenance.jsp page.
The Add New and Change buttons on the inventProdMaintenance.jsp forwarded the user to the inventoryAdmin.jsp. The Change button forwards the user with the selected inventory/product crust object from the list to the inventProdAdmin.jsp page. The inventProdAdmin.jsp page gives the user a way to add a new inventory/product crust object or make changes to the select inventory/product crust object in the application. The inventProdAdmin.jsp displays variable fields, Inventory ID, Product ID, Size, Inventory Description, and Item Message. There are also Inventory Product Active option fields to set the item active or inactive. The Inventory ID field allows the user to select an inventory object that is going to be associated to a product item object. The Product ID field is used to select a product crust object that is going to be associated to the inventory object. The Size field allows the user to select what size is associated with the inventory/product crust object. The Inventory Product Description field allows the user to give a description of the inventory/product crust object for displaying across the application. The Inventory amount allows the user to set the amount of the inventory that is reduced when a product crust is sold. For the Add New item, the inventProdAdmin.jsp page displays a Next and a Cancel button across the top. For the Change item, the inventoryAdmin.jsp displays an Update and a Cancel button across the top of the inventoryAdmin.jsp page and the Inventory ID field is visible but not editable. When the user clicks on the Next button, a new inventory/product crust object is added to the application. When the user clicks on the Update button, the changes to the select inventory/product crust object are updated in the application. When the user clicks on the Cancel button, the user is forwarded back to the inventProdMaintenance.jsp. Figure 8.21 is screenshots of the inventProdAdmin.jsp page.

8.4 Employee Maintenance

When the user clicks on the Employee link from the adminMain.jsp page, the user is forwarded to the employeeMaintenance.jsp page. The Employee Maintenance page gives the user a way to maintain the employees in the application. The page displays a list of
employees in the application. The list allows the user to view all the employees that exist in the application. The list allows the user to select an employee to remove from the application, or makes change to information in the application. The employeeMaintenance.jsp page has: Add New, Change, Delete and Admin Menu buttons placed across the top of list. The Admin Menu button forwards the user back to the adminMain.jsp page. The Delete button removes the selected employee from the application. The Add New button allows the user to add a new employee to the application. The Change button allows the user to change the employee’s information. Figure 8.22 is a screenshot of the employeeMaintenance.jsp page.

<table>
<thead>
<tr>
<th>Employee ID</th>
<th>Employee Name</th>
<th>Active</th>
<th>In-Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDWARD</td>
<td>Edward Gerhardstein</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>JOHN</td>
<td>John Smith</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.22: Screenshot of the employeeMaintenance.jsp page.

The Add New and Change buttons on the employeeMaintenance.jsp page forwards the user to the employeeAdmin.jsp page. The Change button forwards the user with the selected employee from the list to the employeeAdmin.jsp page. The employeeAdmin.jsp page gives the user a way to add a new employee or make changes to the selected employee in the application. The employeeAdmin.jsp displays variable fields: Employee ID, First Name, Last Name, Address, City, State, Zip, and SSN. There are also Employee Active option fields to set the employee active or inactive. The Employee ID field is to give a unique identifier to an employee. When a user tries to add a new employee, if an Employee ID already exists in the application, the user is redirected back to the employeeAdmin.jsp page with an error message. The First Name field is used to fill in the first name of the employee. The Last Name field is used to fill in the last name of the employee. The Address field is used to add the street address of the employee. The City field is used to enter the city of the employee’s address. The State field is used to enter the state of the employee’s address. The Zip field allows the user to provide a zip code of the employee’s address. The SSN field allows the user to enter the employee’s Social Security Number. For Add New employee, the employeeAdmin.jsp displays a Next and a Cancel button across the top. For the Change, employee the employeeAdmin.jsp page displays an Update and a Cancel button across the top of the employeeAdmin.jsp page and the Employee ID field is visible but not editable. When the user clicks on the Next button on the employeeAdmin.jsp, a new employee is added to the application. When the user clicks on the Update button on the employeeAdmin.jsp the employee’s information is updated in the application. When the user clicks on the Cancel button on the employeeAdmin.jsp the user is forwarded back to the employeeMaintenance.jsp. Figure 8.23 is screenshots of the employeeAdmin.jsp page.
8.5 User Maintenance

When the user clicks on the User link from the adminMain.jsp page, the user is forwarded to the userMaintenance.jsp page. The User Maintenance page gives the user away to maintain the users in the application. The userMaintenance.jsp page displays a list of users in the application. The list allows the user to view all the users that exist in the application. The list also allows the user to select a user to remove from the application or change the user’s information in the application. The userMaintenance.jsp page has: Add New, Change, Delete and Admin Menu buttons placed across the top of list. The Admin Menu button forwards the user backed to the adminMain.jsp page. The Delete button removes the selected user from the application. The Add New button allows the user to add a new user to the application. The Change button allows the user to make changes to the user’s information. Figure 8.24 is a screenshot of the userMaintenance.jsp page.
The Add New and Change button on the userMaintenance.jsp forwarded the user to the userAdmin.jsp. The Change button forwards the user, with the selected user from the list, to userAdmin.jsp page. The userAdmin.jsp page gives the user a way to add a new user or make changes to the selected user in the application. The userAdmin.jsp displays variable fields, User ID, First Name, Last Name, Password, Confirm Password, and Employee ID. There are also User Active, User Order Taker, User Supervisor, User Administrator, User Dispatcher, and User Authorize option fields. The User ID field allows user the set a unique identifier to a new user and the User ID is also used by the user to logon the application. When adding a new user, if the User ID already exists in the application, the user is forwarded back the userAdmin.jsp with an error message. The First Name field allows the user to enter the user's first name. The Last Name field allows the user to enter the user's last name. The Password field is used to enter a password for the user. The Confirm Password field is used to reenter a password for the user. If both the Password field and Confirm Password field doesn’t match, the user is forwarded back to the user-Admin.jsp with an error message. The Employee ID field allows the user to associate an employee to the User ID. The User Active option field allows the user to set a user to be active or inactive. The User Order Taker option field allows the user to have access to the Order Functionality. The User Supervisor option field allows the user to have access to the Manager Functionality. The User Administrator option field allows the user to have access to the Administrator Functionality. The User Dispatcher option field allows the user have access to the Dispatch/Cashier Functionality. The User Authorize option field is to set the user as an authorized user. For the Add New user, the userAdmin.jsp displays a Next and a Cancel button across the top. For the Change user, the userAdmin.jsp page displays an Update and a Cancel button across the top of the userAdmin.jsp page and the User ID field is visible but not editable. When the user clicks on the Next button on the userAdmin.jsp page, a new user is added to the application. When the user clicks on the Update button on the userAdmin.jsp the user information is updated in the application. The Cancel button on the userAdmin.jsp forwards the user back to the userMaintenance.jsp. Figure 8.25 is screenshots of the userAdmin.jsp page.
Figure 8.25: Screenshots of the userAdmin.jsp page
9.1 Management Functionality Overview

The Management Functionality was designed to give the user a way to manage the business. The Management Functionality gives the user the capability to manage employee’s schedules, manage inventory and run reports. The manager user role gives the user access to the Management Functionality.

When the user clicks on the Management link from the main.jsp page, the user is forwarded to the Management Main entry page, manageMain.jsp. The manageMain.jsp page contains a list of Management hyperlinks. The list contains links to Reports, Schedules, and Inventory. Like the Administrator functionality, the Management functionality is divided up into groups, and each management task is placed in these groups based on how closely the tasks are associated to the group. Dividing the management task groups makes it easier for the user to navigate through the application to get to the task they want to perform. The Reports hyperlink forwards the user to the Report Main entry page, reportMain.jsp. The Schedule hyperlink forwards the user to Schedule Management entry page, scheduleMain.jsp. The Inventory hyperlink forwards the user to the Inventory Management Main entry page, inventMain.jsp. Figure 9.1 is a screen shot of the manageMain.jsp page.

<table>
<thead>
<tr>
<th>Pizza Restaurant</th>
<th>Management Main</th>
<th>Pizza Application</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Management Main:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>➤ Reports</td>
<td>Create and View Reports</td>
<td></td>
</tr>
<tr>
<td>➤ Schedule</td>
<td>Create Employees Schedule</td>
<td></td>
</tr>
<tr>
<td>➤ Inventory</td>
<td>Adjust Inventory</td>
<td></td>
</tr>
<tr>
<td>➤ Main</td>
<td>Back to Main</td>
<td></td>
</tr>
</tbody>
</table>

Figure 9.1: Screenshot of the manageMain.jsp page.

9.2 Schedule Management

When the user clicks the Schedule link from the manageMain.jsp page, the user is forwarded to the scheduleMain.jsp page. The Schedule Management page gives the user a way to create and view employee’s schedules. When first entering the scheduleMain.jsp page,
the scheduleMain.jsp page displays a list of employees and their current weekly schedule. The list is comprised of all employees, regardless if the employee has hours scheduled or not. This makes it easier for the manager to view all the employees scheduled hours and select an individual employee to edit his hours for the week. The scheduleMain.jsp page also has buttons for: Current Week, Next Week, Second Week, Prev Week, Edit, and Manager Main. Figure 9.2 is a screen shot of the scheduleMain.jsp page.

![Management Main Menu](image)

Figure 9.2: Screenshot of the scheduleMain.jsp page.

The Current Week button updates the employee schedule list with the current week schedule. The Next Week button updates the employee schedule list with the next week schedule. The Next Week button gives the user or manager a way to create and view the schedule for the employees for a week in advance from the current week. The Second Week button updates the employee schedule list with a schedule for the two week after the current week. The Second Week button gives the user or manager a way to create and view the schedule for the employees for two weeks in advance of current week. The Prev Week button updates the employee schedule list with the previous week schedule. The Prev Week button gives the user or manager a way to view the previous week schedule of employees. This can help the manager to view a schedule of the employees, and use it to create a new schedule for current or future weeks. The Manager Main button forwards the user back to the manageMain.jsp page. The Edit button forwards the user or manager with the selected employee to empSchedule.jsp page and allows the manager to edit the selected employee schedule.

The empSchedule.jsp page was designed to give the user or manager a way to change the selected employee schedule for a particular week. The empSchedule.jsp page displays the employee’s name and have buttons for Save and Cancel. The empSchedule.jsp page also has variables for time in and out for each day of the week, from Sunday to Saturday. The empSchedule.jsp page also has option variables for AM/PM for in and out for each day of the week, from Sunday to Saturday. When the user clicks on the Cancel button, the user is forwarded back to the scheduleMain.jsp page. When the user clicks on the Save button
the employee’s schedule is saved and the user is forwarded back empSchedule.jsp page.

9.3 Inventory Management

When the user or manager clicks on the Inventory link from the manageMain.jsp page, the user is forwarded to the inventMain.jsp page. The Inventory Management allows the user to manage inventory in the web application to match the local store stock. There are two things a user can do with the Inventory, add new inventory when inventory is delivered to the store, and edit the inventory when the inventory in the store does not match the inventory in the application. The inventMain.jsp page was designed to give the user a way to select which task he wants to perform. The inventMain.jsp consists of a list of inventory hyperlinks. The list contains links to: Inventory Delivery, Edit Inventory, Management Main, and Main. The Management Main and Main links forwarded the user back to the application manageMain.jsp and main.jsp pages respectively. The Inventory Delivery hyperlink is linked to the Inventory Delivery page, inventDelivery.jsp. The Edit Inventory hyperlink is linked to the Inventory Item Management page, inventManagement.jsp. Inventory Management allows the user to add inventory items that were delivered to the store, into the application. It also allows the user to edit the inventory on hand. One of the reasons the user needs to edit the inventory on hand, is for shrinkage. Shrinkage is a term used to account for loss of products. The primary reason for shrinkage is due to spoilage. Figure 9.3 is a screen shot of the inventMain.jsp page.

<table>
<thead>
<tr>
<th>Pizza Restaurant</th>
<th>Inventory Management Main</th>
<th>Pizza Application</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inventory Management Main:</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| ![List of hyperlinks](image)

Figure 9.3: Screenshot of the inventMain.jsp page.

9.4 Inventory Delivery

When the user clicks on the Inventory Delivery link from the inventMain.jsp page, the user is forwarded to the inventDelivery.jsp page. The Inventory Delivery page gives the user a way to add inventory that has been delivered to the store. The inventDelivery.jsp page displays a list of inventory delivered, this list should be empty when forwarded from inventMain.jsp page. When there are items in the list, the user can view, edit and add
new inventory items that have been delivered. The inventDelivery.jsp page has: Add New, Change, Delete, Save and Inventory Main buttons placed across the top of the list. When a user clicks on the Inventory Main button, the user is forwarded back to the inventMain.jsp page. When a user clicks on the Remove button, the selected inventory item is removed from the list of inventory delivered. When a user clicks on the Save button, the list of inventory item delivered is save into the database and add to the inventory quantity on hand. The Add New button gives the user a way to add a new inventory item delivered. The Change button gives the user a way to edit a selected inventory item delivered. Figure 9.4 is a screen shot of the inventDelivery.jsp page.

![Inventory Delivery](image1)

Figure 9.4: Screenshot of the inventDelivery.jsp page.

The Add New and Change buttons on the inventDelivery.jsp page forwards the user to the inventoryDeliveryItem.jsp page. The Change button forwarded the use with the selected inventory item from the list on inventDelivery.jsp page to inventoryDeliveryItem.jsp page. The inventoryDeliveryItem.jsp page gives the user a way to add a new inventory item to the delivery list. The inventoryDeliveryItem.jsp page also gives the user a way to change inventory item in the delivery list. The inventoryDeliveryItem.jsp displays variable fields for Inventory ID, Inventory Amount, Inventory Cost, and Delivery Date. The Inventory ID field gives the user a way to select the inventory item that is delivered. The Inventory Amount field allows the user to enter the total amount of the inventory item delivered. The Inventory Cost field allows the user to enter the total cost of the inventory item delivered. The Delivery Date field is used to enter a date the inventory item was delivered. For the Add New item, the inventoryDeliveryItem.jsp page displays a Next and a Cancel button across the top. For the Change item the inventoryDeliveryItem.jsp page displays an Update and a Cancel button across the top and the Inventory ID field is visible but not editable. When the user clicks on the Next button, a new inventory delivery item is added to the delivery list and is forwarded back to inventDelivery.jsp page. When the manager clicks on the Update button, the changes made on select inventory delivery item are updated in the delivery list and is forwarded back to inventDelivery.jsp. The Cancel button, forwards the user back to the inventDelivery.jsp. Figure 9.5 is screenshots of the inventoryDeliveryItem.jsp page.

### 9.5 Inventory Delivery

When the user clicks on the Edit Inventory link from the inventMain.jsp page, the user is forwarded to the Inventory Item Management page, inventManagement.jsp. The Inventory
Figure 9.5: Screenshot of the inventoryDeliveryItem.jsp page.

Item Management page gives the user a way to edit the inventory on hand. The invent-Management.jsp page displays a list of inventory items and the total number of inventory items on hand. The inventManagement.jsp page was designed to allow the user to view all the inventory items and the amount that is supposed to be in the store. If the amount is different from what is in the store, the user can select that inventory item and change it. The inventManagement.jsp page has, Work With and Inventory Main buttons placed across the top of list. When the user clicks on the Inventory Main button, the user is forwarded the back to the inventMain.jsp page. Figure 9.6 is screen shot of the inventManagement.jsp page.

Figure 9.6: Screenshot of the inventManagement.jsp page.

When the user clicks on the Work With button on the inventManagement.jsp page, the user is forwarded with the selected inventory item to the inventoryEdit.jsp page. The inventoryEdit.jsp page is used to change the inventory currently on hand. The inventoryEdit.jsp page displays the Inventory ID, Inventory Description and a variable field for Inventory Amount. The inventoryEdit.jsp page also displays Update and Cancel buttons across the top of the page. The Inventory ID and Inventory Description fields are to displaying the
inventory item and description of the inventory item, which allows the user to make sure the correct inventory item was selected. The Inventory Amount field is used to set the correct amount in the application. When the user clicks on the Cancel button, the user is forwarded back to inventManagement.jsp page. When the user clicks the Updates button, the inventory amount is updated and the user is forwarded to the inventManagement.jsp page. Figure 9.7 is screen shot of the inventoryEdit.jsp page.

![Pizza Restaurant Edit Inventory Pizza Application](image)

<table>
<thead>
<tr>
<th>Update</th>
<th>Cancel</th>
</tr>
</thead>
</table>

**Update Inventory On Hand:**

<table>
<thead>
<tr>
<th>Inventory ID:</th>
<th>LGPNCRST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inventory Description:</td>
<td>Large Pan Crust</td>
</tr>
<tr>
<td>Inventory Amount:</td>
<td>30</td>
</tr>
</tbody>
</table>

Figure 9.7: Screenshot of the inventoryEdit.jsp page.

## 9.6 Report Management

When the user clicks on the Inventory link from the manageMain.jsp page, the user is forwarded to the reportMain.jsp page. The Report Management page allows user to view different order activities through the selected reports on the application. The reportMain.jsp page consists of a list of report hyperlinks. The list contains links to: Inventory Report, Driver Report, Employee Report, Product Report, Order Type Report, Management Main, and Main. The Management Main and Main links forwarded the user back to the manageMain.jsp and main.jsp pages respectively. The Inventory Report hyperlink is linked to the Inventory Report entry page, inventReportStart.jsp. The Driver Report hyperlink is linked to the Driver Report entry page, driverReportStart.jsp. The Employee Report hyperlink is linked to the Employee Report entry page, employeeReportStart.jsp. The Product Report hyperlink is linked to the Product Report entry page, productReportStart.jsp. The Order Type Report hyperlink is linked to the Order Type Report entry page, orderReportStart.jsp. Figure 9.8 is a screen shot of the reportMain.jsp page.

### 9.6.1 Inventory Report

When the user clicks on the Inventory Report hyperlink on the reportMain.jsp, the user is forwarded to Inventory Report Start page, inventoryReportStart.jsp. An Inventory Report is a report that displays the amount of inventory that the store uses during a certain time period. The inventoryReportStart.jsp page has a Start Date and End Date variable fields.
The Start Date and End Date field are used by the user to enter the dates to run a report. The inventoryReportStart.jsp page also has a Next and Cancel button across the top of the page. When the user clicks the Cancel button, the user is forwarded back to the reportMain.jsp. When the user clicks the Next button, the user is forwarded to the Driver Report Display page, inventoryReportDisplay.jsp with a start date and an end date.

When the user is forwarded to the inventoryReportDisplay.jsp page, an inventory report list is generated with the start and end dates. The inventoryReportDisplay.jsp page displays the report list with the Inventory ID, Description, Amount, and Cost. The Inventory ID and the Description fields display the inventory item information. The Amount field displays the total amount sold of the inventory item. The Cost field displays the total cost of the inventory item. The inventoryReportDisplay.jsp page also has a Back and a Cancel button placed across the top of the page. When the user clicks on the Cancel button, the user is forwarded to the reportMain.jsp. When the user clicks on the Back button, the user is forwarded to the inventoryReportStart.jsp.

### 9.6.2 Driver Report

When the user clicks the Driver Report hyperlink on the reportMain.jsp, the user is forwarded to Driver Report Start page, driverReportStart.jsp. A Driver Report is a report that displays the totals on the orders that the drivers delivered during a certain time period. The driverReportStart.jsp page has a Start Date and End Date variable fields. The Start Date and End Date fields are used by the user to enter the dates to run a report. The driverReportStart.jsp page also has a Next and Cancel button placed across the top of the page. When the user clicks the Cancel button, the user is forwarded back to the reportMain.jsp. When the user clicks the Next button, the user is forwarded to the Driver Report Display page, driverReportDisplay.jsp.

When the user is forwarded to driverReportDisplay.jsp page a driver report list is gener-
ated with the start and end dates. The driverReportDisplay.jsp page displays the report list with Driver ID, Hours, Total Orders, and Ave Orders Per Hour fields. The Driver ID field displays the id of a particular driver. The Hours field displays the total hours the driver worked. The Ave Order Per Hour field displays the average numbers of orders the driver delivered per hour. The driverReportDisplay.jsp page also has a Back and a Cancel button place across the top of the page. When the user clicks on the Cancel button, the user is forwarded to the reportMain.jsp page. When the user clicks on the Back button, the user is forwarded to the driverReportStart.jsp.

9.6.3 Employee Report

When the user clicks the Employee Report hyperlink on the reportMain.jsp, the user is forwarded to Employee Report Start page, employeeReportStart.jsp. The Employee Report is used to generate a report list of the total hours an employee works between a given time period. The employeeReportStart.jsp is used to enter a start date and end date to run the report. The employeeReportStart.jsp page has a Start Date and End Date variable fields. The employeeReportStart.jsp page also has a Next and Cancel button place across the top of the page. When the user clicks the Cancel button, the user is forwarded back to the reportMain.jsp. When the user clicks the Next button, the user is forwarded to the Employee Report Display page, employeeReportDisplay.jsp.

When the user is forwarded to employeeReportDisplay.jsp page, an employee report list is generated for the start and end dates. The employeeReportDisplay.jsp page displays the list of Employee ID, Name, and Total Hours fields. The Employee ID field displays the unique identifier assigned to the employee. The Name field displays the full name of the employee. The Total Hours field displays the total hours the employee worked. The employeeReportDisplay.jsp page also has a Back and a Cancel button place across the top of the page. When the user clicks on the Cancel button, the user is forwarded back to the reportMain.jsp. When the user clicks on the Back button, the user is forwarded to the employeeReportStart.jsp.

9.6.4 Product Report

When the user clicks the Product Report hyperlink on the reportMain.jsp, the user is forward to Product Report Start page, productReportStart.jsp. The Product Report is used to generate a report list of the totals for each product sold between a given time period. The productReportStart.jsp is used to enter a start date and end date to run the report. The productReportStart.jsp page has a Start Date and End Date variable fields. The productReportStart.jsp page also has a Next and Cancel button place across the top of the page. When the user clicks the Cancel button, the user is forwarded back to the reportMain.jsp. When the user clicks the Next button and the user is forwarded to the Product Report Display page, productReportDisplay.jsp.

When the user is forwarded to productReportDisplay.jsp, a product report list is generated for the start and end dates. The productReportDisplay.jsp page displays the list of Product ID, Product Desc, and Total Sold fields. The Product ID and Product Desc fields display the product identification and the description of the product. The Total Sold field displays
the total sold of the product between the start and end dates. The productReportDisplay.jsp page also has a Back and a Cancel button place across the top of the page. When the user clicks on the Cancel button, the user is forwarded to the reportMain.jsp. When the user clicks on the Back button, the user is forwarded to the productReportStart.jsp.

9.6.5 Order Type Report

When the user clicks on the Product Report hyperlink on the reportMain.jsp, the user is forwarded to Order Type Report Start page, orderTypeReportStart.jsp. The Order Type Report is used to generate a report list of the totals for each order type (Dine-in, Carry-out, and Delivery) sold between a given time period. The orderTypeReportStart.jsp is used to enter a start date and end date to run the report. The orderTypeReportStart.jsp page has a Start Date and End Date variable fields. The orderTypeReportStart.jsp page also has a Next and Cancel button placed across the top of the page. When the user clicks the Cancel button, the user is forwarded back to the reportMain.jsp. When the user clicks the Next button, the user is forwarded to the Order Type Report Display page, orderTypeReportDisplay.jsp.

When the user is forwarded to orderTypeReportDisplay.jsp page, order type report list is generated for the start and end dates forwarded. The orderTypeReportDisplay.jsp page displays the list of Type ID, Type Desc, and Total Sold fields. The Type ID and Type Desc, fields display the identification code associated to the order type and the description of the order type. The Total Sold field displays the total orders sold for each particular order type. The orderTypeReportDisplay.jsp page also has a Back and a Cancel button placed across the top of the page. When the user clicks on the Cancel button, the user is forwarded to the reportMain.jsp. When the user clicks on the Back button, the user is forwarded to the orderTypeReportStart.jsp.
Chapter 10
Order Functionality

10.1 Order Functionality Overview

The Order Functionality was designed to give the user the ability to take and change orders for dine-in, delivery, and carry out. The whole purpose of a restaurant and this application is to take orders from customers. Taking and fulfilling orders is how a restaurant makes money. The Order Functionality section of this application gives the restaurant the ability to do that. To access the Order Functionality, the user must have the order taker role.

When the user clicks on the Order link from the main.jsp page, the user is forwarded to the Order Main page, orderMain.jsp. The orderMain.jsp page contains a list of Order hyperlinks. The list contains a link to: Dine-In, Delivery, Carry Out, and Change Order. The orderMain.jsp page was designed to allow the user to select the type of order the user wants to take and also make changes to an order already taken. The Dine-In, Delivery, and Carry Out hyperlinks are linked to the Order Entry page, orderEntry.jsp. When the user clicks on the Dine-In, Delivery, and Carry Out hyperlink, the type of order is assigned and is forwarded to the orderEntry.jsp page. The Change Order hyperlink is link to the Order List page, orderEdit.jsp. All the links have similar processes of taking and changing orders, accept for the first few JSP pages. Figure 10.1 is a screenshot of the orderMain.jsp page.

In the Management and Administration sections of the application, the web pages gather information from one xxx.jsp page and send it to the XXXAction Class. The XXXAction Class calls a method in the XXXManagementBean to make a database connection and manipulate the data in the database. It is usually a one-to-one mapping of data to the xxx.jsp
The Order section of the application on the other hand, must collect data across multiple JSP pages and Action Classes. For this reason, a container object was created to store the data from each JSP page and Action Class along the way. Each JSP page and Action Class takes the container, adds its own little pieces to the container, and then passes the container to the next JSP page and Action Class. The OrderBean and the OrderItemBean container where created to store the order data. Since each order can have more than one order item, the OrderItemBean container was created to store data about that particular order item. The OrderBean container, on the other hand, was created to store data about the order, including a list of OrderItemBeans. Figure 10.2 is a diagram of the order process.

Figure 10.2: Diagram of the order process.

The Order section could have been designed with everything about the order on one single page, but that would have made the page too busy looking. It also would have made it difficult to develop and maintain. The idea behind the design of the Order section was to make it easier for the user to follow.

10.2 Order Entry Page

When the user clicks on the Dine In hyperlink, the user is forwarded to the Order Entry page, orderEntry.jsp. The orderEntry.jsp page is an entry point to take an order. When
taking a Dine In order, the orderEntry.jsp page is used to assign a table number and server to the order. The orderEntry.jsp page also has a Submit and Cancel button on the page. When the user clicks the Cancel button, the user is forwarded back to orderMain.jsp page. When the user clicks on the Submit button, the information is saved in the OrderBean and the user is forwarded to the Item Type page, orderItem.jsp. Figure 10.3 is a screenshot of the Dine In orderEntry.jsp page.

![Screenshot of the Dine In orderEntry.jsp page.](image)

When the user clicks on the Delivery hyperlink, the user is forwarded to the Order Entry page, orderEntry.jsp. The orderEntry.jsp page is an entry point to take an order. When taking a Delivery order the orderEntry.jsp page has variable fields for the Phone Number and Name. The Phone Number and Name fields are used to take phone number and name of the customer. The orderEntry.jsp page also has a Submit and Cancel buttons on the page. When the user clicks the Cancel button, the user is redirected back to orderMain.jsp page. When the user clicks on the Submit button, the information is saved in the OrderBean and the user is forwarded to the Order Customer Address page, orderAddress.jsp. Figure 10.5 is a screenshot of the Delivery and Carry Out orderEntry.jsp page.

The orderAddress.jsp page is used to gather the customers address to deliver the order to. The customer name and phone number is displayed at the top of the page. If the customer had a previous delivery with the phone number entry, an address drop down menu should be populated. The user can either select an address from a drop down menu or enter a new customer address in the address variable fields. The address variables fields on the page are: Address1, Adress2, City, State, Zip and Comment. The Address1 field is used to enter the customer’s street address. The Address2 field is used to enter additional address information if necessary. The City, State, and Zip fields are used to enter the city, state, and zip code of the customer’s address. The Comment field is used to take any special instructions the customer wants to give on the address. The orderAddress.jsp page also has a Submit and Cancel button on the page. When the user clicks the Cancel button, the user is
redirected back to the orderMain.jsp page. When the user clicks on the Submit button, the customer’s address information is saved in the OrderBean and the user is forwarded to the Item Type page orderItem.jsp. Figure 10.4 is a screen shot of the Delivery orderAddress.jsp page.

![Order Customer Address](image)

Figure 10.4: Screenshot of the orderAddress.jsp page.

When the user clicks on the Carry Out hyperlink, the user is forward to the Order Entry page, orderEntry.jsp. The orderEntry.jsp page is an entry point to take an order. When taking a Carry Out order the orderEntry.jsp page has variable fields for the Phone Number and Name. The Phone Number and Name fields are used to enter the customer name and phone number. The orderEntry.jsp page also has a Submit and Cancel button on the page. When the user clicks on the Cancel button, the user is redirected back to the orderMain.jsp page. When the user clicks on the Submit button, the customer’s information is saved in the OrderBean and the user is forwarded to the Item Type page orderItem.jsp. Figure 10.5 is a screenshot of the Delivery and Carry Out orderEntry.jsp page.

10.3 Order Item Type

The Item Type page, orderItem.jsp, has a list of product categories: Pizza, Pasta, Sandwich, Salad, Side, Beverage, and Alcohol. The orderItem.jsp page was designed to allow a user to view all the order item types and select an order item type to add to the order. When the user clicks on one of the Item Type categories, an OrderItemBean is created with the category type, and the user is forwarded to the Order Item JSP page, orderItem.jsp. Figure 10.6 is a screen shot of the orderItem.jsp page.

10.4 Order Item

When the user clicks on one of the categories from the list on the orderItem.jsp page the user is forwarded to the Order Item page, orderItem.jsp. Depending on which Item Type category the user chooses, the user is forwarded to one of three different Order Item
pages. The first Order Item page, orderItemsType1.jsp, has a Size, Type, and Product drop down menu. This is where AJAX is implemented. When the user selects the Size drop down menu, it calls two JavaScript functions. The two JavaScript functions then call two Java Servlets. The Java Servlets then pull information from the database for the Type and Product drop down menus and passes that information back to the JavaScript functions. The JavaScript functions then populate the Type and Product drop down menus. This is all done without needing to reload the page. The user can now choose the order item type and product. There are also a Cancel and a Next button across the top of the page. When the user clicks the Cancel button, the user is forwarded back to orderMain.jsp page. When the user select a Size, Type, and Product, and clicks on the Next button, the data is saved in the OrderItemBean and user is forwarded to the Order Item Topping page, orderItemsType1p2.jsp. Figure 10.7 is a screen shot of the orderItemsType1.jsp page.

On the Order Item Topping page, there is a list of toppings or extras that the user can choose from. The user can select the requested topping by clicking on the name of the topping. The user clicks the name once and a red “+” will show up next to the name. By clicking the name of the topping two times, a red “x” will show up next to the name, signifying that the user wants extra of that topping. Clicking the name three times will show a red “-” next to the name signifying that the user wants to remove that item from the pizza. For instance, if the user wants a meat lover’s pizza with no sausage, the user would click on the sausage name three times. By clicking the name four times the space is clear next to the name. A JavaScript function was created to keep track of how many times the item has been clicked and then update the space next to the name with the appropriate symbol. There are also a Cancel and Next button placed across the top of the page. When the user clicks the Cancel button, the user is forwarded back to orderMain.jsp page. When the user clicks the Next button, the toppings selected are saved to the OrderItemBean and the user is forwarded to the Order Item Summary page, orderSum.jsp. The figures below are screen shots of the orderItemsType1p2.jsp page. Figure 10.8 is screenshots of the orderItemsType1p2.jsp page.
### 10.5 Order Sum

When the user is forwarded to the Order Item Summary page, orderSum.jsp, the user can view the summary of the entire order. The Order Summary displays the name, address and phone number of the customer. It also shows a list of all the items ordered, along with the price of each item, and total price of the order. The orderSum.jsp page also displays a list of buttons across the top. The list of buttons gives the user the option to Save, Add Item, Change Item, Remove, and Add Credit Card. The Add Item button gives the user the ability to add more items to the order. The Remove button allows the user to remove items from the order. The Change Item button allows the user to change items on the order. Figure 10.9 is a screen shot of the orderSum.jsp page.

The Add Credit Card button allows the user to authorize a credit card for the order. When the user clicks on the Add Credit button, the CreditCard JSP page pops up. The CreditCard JSP page is used to take the credit card information and get credit card approval. This is done through propriety software of the credit card machine bought by the individual restaurant. The software has to be installed on the server and is called from application when credit card approval is needed. The software takes the credit card information either by manually entering, or captured through a card swipe device. Once the credit information is taken, it gets authorized by the software and returns the authorization code. The software can either input the authorization code directly into the application database, or just return the authorization code to the application. The application then takes the authorization code and associates it with the order in the database. The application does not need to store the credit card information anywhere in the database. The authorization code is all the business needs to collect the credit card charge.

When the user clicks on the Save button, there are several things that can happen depending...
on if the order is a new order or a changed order.

If the order is a new order, the application creates a new entry in the Orders table, and creates new entries in the OrdItem, and OrdExtra tables. The entries in the OrdItem are associated with the Order ID in the Orders table. The new entry in the OrdExtra table is associated with the OrdItem ID in the OrdItem table and Order ID in the Orders table. The OrdExtra table stores the toppings or any extra products on the OrderItem.

If the order is a changed order, the application updates the entry associated with the Order ID in the Order table. The application removes the entries from the OrdItem and OrdExtra tables that are associated with the Order ID. The application then adds new entries in the OrdItem, and OrdExtra tables the same way as above.

Next, the application updates to the Inventory on Hand tables. It does this by retrieving the inventory item quantity amount for all product items, product crust, and product extras. This is done based on the type of product and size.

The last thing the application does is to add a new entry or update the OrderCredit table with the credit card approval number and order amount. After all the information about the order is updated and inserted in to the database, the user is forwarded to the orderMain.jsp page.

10.6 Order Change

The Change Order link from the Order Main page forwards the user to the Order List page, orderEdit.jsp. The orderEdit.jsp was designed to view all orders and to make changes to or cancel a selected order. The orderEdit.jsp was also designed to give the user a way to choose different views of orders. The Order List page displays a list of orders the user can select. The Order List page also displays several rows of buttons across the top. The
Open Orders, Delivery Orders, Carry-Out Orders, Dine-In Orders, and Canceled Orders buttons provide different viewing options for the user to view certain orders. The Open Orders button allows the user to view all open orders. The Delivery Orders button allows the user to view all delivery orders. The Carry-out Orders button allows the user to view all carry out orders. The Dine-in Orders button allows the user to view all dine in orders. The Canceled Orders button allows the user to view all canceled orders. The All Orders button allows the user to view all the orders. There are also three more buttons the Change, Cancel Order and Cancel button. When the user clicks on the Cancel button, the user is redirected back to the orderMain.jsp page. The Cancel Order button cancels the select order. When the user clicks on the Change Order button, the user is forwarded to the Order Item Summary page, orderSum.jsp, with the selected order. Figure 10.10 below is a screenshot of the orderEdit.jsp page.
Figure 10.9: Screenshot of the Order Summary page.

<table>
<thead>
<tr>
<th>Restaurant</th>
<th>Order Item</th>
<th>Pizza Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save</td>
<td>Add Item</td>
<td>Change Item</td>
</tr>
<tr>
<td>Remove Item</td>
<td>Credit Card</td>
<td>Cancel</td>
</tr>
</tbody>
</table>

Order Summary

<table>
<thead>
<tr>
<th>Order Numbers</th>
<th>Order List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td>Product Main:</td>
</tr>
<tr>
<td>Address:</td>
<td>Order ID</td>
</tr>
<tr>
<td></td>
<td>12 31062556589</td>
</tr>
<tr>
<td></td>
<td>11 1</td>
</tr>
<tr>
<td></td>
<td>10 31062556589</td>
</tr>
<tr>
<td></td>
<td>9 31062556589</td>
</tr>
<tr>
<td></td>
<td>8 1</td>
</tr>
</tbody>
</table>

Figure 10.10: Screenshot of the orderEdit.jsp page.
11.1  Dispatch /Cashier Functionality Overview

The Dispatch/Cashier Functionality was designed to give the user the ability to finalize orders. This includes cashing out orders and dispatching delivery orders. It is divided into two different parts, one for dispatching delivery orders, and one for cashing out dine-in and carry out orders.

When the user clicks on the Dispatch/Cashier link from the Main Menu page, the user is forwarded to the Dispatch/Cashier Main page, dispCashMain.jsp. The Dispatch/Cashier Main page has three hyperlinks: Dispatch, Cashier and Main. The Main hyperlink forwards the user back to the Main Menu page. Figure 11.1 below is a screenshot of the dispCashMain.jsp page.

![Screenshot of the dispCashMain.jsp page.](image)

11.2  Cashier Main

The Cashier link on the Dispatch/Cashier page forwards the user to the Cashier Main Menu page, cashierMain.jsp. The cashierMain.jsp page was designed to cash out dine-in and carry out orders. On the Cashier Menu page, there is a list of dine-in and carry out orders. There is also a Close and CashOut Order button placed across the top of the page. When the user clicks on the Close button, the user is redirected back to the Dispatch/Cashier page. When the user clicks on the CashOut Order button, the user is forwarded to the Cashout Order page, cashoutOrder.jsp, with the selected order to cash out.

The Cashout Order page gives the user the ability to cash out an order. The cashoutOrder.jsp
page displays the order number, the total price of the order, the credit card amount, the total due, and change due. There is also a Total Paid variable field to enter paid amount. There are three buttons displayed across the bottom of the page. When the user clicks on the Close button, the user is forwarded back to the Cashier Main Menu page. When the user enters the amount in the Total Paid variable field and click on the Submit button, the order is set to close and is forwarded back to the Cashout Order page with Change Due updated. Just like the Order Item Summary page, the user has an option to pay for the order with a credit card. After getting the credit card information, the user is forwarded back to the Cashout Order page with the credit card amount. The Credit Card Amount, the Total Due and the Change Due values are updated with the credit card amount.

11.3 Dispatch Main

When the user clicks the Dispatch link on the dispCashMain.jsp page, the user is forwarded to the dispatchMain.jsp. One of the things a user needs to dispatch an order for delivery are drivers to deliver the orders. The Dispatch section must not only have the ability to assign orders and reassign orders to drivers, but must also have the ability to perform other tasks that is associated with the drivers. The dispatchMain.jsp page gives the user the ability to perform tasks that deal with dispatching orders and other tasks that are associated with drivers. The dispatchMain.jsp page consists of a list of hyperlinks: Dispatch, Driver Bank, Driver In, Cash Out Driver, Reassign Order, Dispatch/Cashier Main, and Main. The Dispatch/Cashier Main and Main links forward the user to the dispCashMain.jsp and the main.jsp pages respectively. The Dispatch hyperlink is linked to the Dispatch Main Menu page, dispatch.jsp. The Driver Bank hyperlink is linked to the Driver Bank page, driver-Bank.jsp. The Driver In hyperlink is linked to Select Driver In page, selectDriveIn.jsp. The Cash Out Driver hyperlink is linked to the Select Drive page, selectDriver.jsp. The Unassigned Orders hyperlink is linked to the Select Drive page, unassignOrders.jsp.
11.3.1 Dispatch Order

The Dispatch link on the Dispatch Main page forwards the user to the Dispatch Main Menu page, dispatch.jsp. The dispatch.jsp page was designed for the user to select one or more orders to dispatch to a driver. On the Dispatch Main Menu page there is a list of pending delivery orders and available drivers. The Dispatch Main Menu page also has Dispatch Order and Dispatch Main button placed across the top. The Dispatch Main button forwards the user back to the dispatchMain.jsp page. The Dispatch Order button allows the user to dispatch the selected orders and assign them to a driver. When the user clicks on the Dispatch Order button, the user is forwarded with the select orders to the Dispatch Order page, dispatchOrder.jsp.

The Dispatch Order page, dispatchOrder.jsp, was designed to allow the user the option to assign multiple orders to a driver to deliver. The page displays a drop down menu to select a driver to assign the orders to. The page was designed to allow the driver to be dispatched with up to five orders at a time. The page can display up to five drop down menus for the user to select orders to assign to the driver. There are also a Save and Cancel button at the bottom of the page. When the user clicks on the Cancel button, the user is forwarded back to the dispatch.jsp page. When the user clicks on the Save button, the selected orders are updated in the database with the selected driver and the order status is set to dispatch. After the database is updated, the user is forwarded back to the dispatch.jsp.

11.3.2 Driver Bank

The Driver Bank was designed to give the user a way to create a driver with a starting bank (cash) amount. When the user clicks on the Driver Bank hyperlink, the user is forwarded to the Driver Bank page, driverBank.jsp. The driverBank.jsp displays the Driver ID and Bank Amount variable fields. The Driver ID field is used to enter a driver identification and is used to be assigned to an order. The Bank Amount is used to enter the starting amount of cash given to a driver for give out change to customers when delivering an order. The page also has a Save and Cancel button on the bottom of the page. When the user clicks on the Cancel button, the user is forwarded back to the dispatchMain.jsp page. When the user clicks on the Save button, a new driver is created with the values entered in the Driver Id and Bank Amount and forwarded back to the dispatchMain.jsp page.

11.3.3 Driver In

The Driver In was designed to give the user a way to cash out orders that the driver has delivered. When the user clicks on the Driver In hyperlink, the user is forwarded to the Select Driver In page, selectDriverIn.jsp. The selectDriverIn.jsp page was designed to select a driver to cash out their orders. The Select Driver In page has a drop menu to select a driver to cash out orders. The Select Driver In page also has a Next and Cancel button across the bottom of the page. When the user clicks on the Cancel button, the user is forwarded back to the dispatchMain.jsp page. When the user clicks on the Next button, the user is redirected to the Driver In page, driverIn.jsp, with the selected driver to cash out their orders.
The driverIn.jsp page was designed to select an individual order of the driver to cash out. The Drive In page display the Driver ID and a list of orders the driver has, with the Order Number and Amount. Each order has a CashOutOrder button displayed next to them. At the bottom of the page the Total Due field is displayed. Also the driverIn.jsp page has a Done and Cancel button across the bottom of the page. When the user clicks on the Cancel button, the user is forwarded back to the dispatchMain.jsp page. When the user clicks on the Done button, and the Total Due is 0, the driver is update to in and the user is forward back to the dispatchMain.jsp page. If the user Total Due is not 0 then the user is forward back to the dispatchMain.jsp page. When the user clicks on the CashOutOrder button next to the order, the user is forwarded to the Cashout Order page. The Cashout Order page is the same Cashout Order page describe in the Cashier section above. When the user is finished cashing out the selected order, the user is forwarded back to the driverIn.jsp page and that order is set to zero. The user keeps on cashing out each order until the total is zero.

11.3.4 Cashout Driver

The Cashout Driver option was designed to give the user a way to cash out the driver’s bank at the end of the driver’s shift. When the user clicks on the Cashout Driver hyperlink on the dispatchMain.jsp page, the user is forwarded to the Select Drive page, select- Driver.jsp. The selectDriver.jsp page was designed to select a driver to cash out for the day. The Select Driver page has a drop menu to select a drive to cash out for the day. The Select Driver page also has a Next and Cancel button two buttons across the bottom of the page. When the user clicks on the Cancel button, the user is redirected back to the dispatchMain.jsp page. When the user clicks on the Next button, the user is forwarded to the Cashout Driver page, cashoutDriver.jsp with the select driver.

The Cashout Driver page displays the Bank Amount, Number of Orders, Commission Rate, Total Due, Total Commission, and an input box for Total Payout. The Bank Amount field displays the bank amount the driver was given at the start of the shift. The Number of Orders field displays the number of orders the driver delivered from the start of the drivers shift. The Commission Rate field displays the current commission rate the driver receives for each order the driver delivers. The Total Commission field displays the commission the driver earned on his shift. The commission is comprised of the total number of orders the driver delivered multiple by the commission rate. The Total Due field displays the bank amount less the total commission the drive earned. If the Total Due is positive, the driver owes the store the Total Due amount. If the Total Due is negative, the store owes the driver the Total Due amount. There also a Save and Cancel button at the bottom of the page. When the user clicks on the Cancel button, the user is redirected back to the dispatchMain.jsp page. When the user clicks on the Next button, the user is forwarded to the Cashout Driver page, cashoutDriver.jsp with the select driver.
11.3.5 Unassigned Orders

The Unassigned Orders option was design to remove the assigned orders that were dispatched to a driver. When the user clicks on the Unassigned Orders hyperlink, the user is forwarded to the Unassigned Order page, unassignOrder.jsp. The unassignOrder.jsp page was designed to select a driver to remove orders that was assigned to the driver. The Unassigned Order page has a drop down menu to select a driver. The Unassigned Order page also has a Next and Cancel two across the bottom of the page. When the user clicks on the Cancel button, the user is forwarded back to the dispatchMain.jsp page. When the user clicks on the Next button, all the orders that are assigned to the driver are unassigned and the Driver is set to in. When the user clicks on the Next button, the user is redirected back to the dispatchMain.jsp page.
### 12.1 Classes Overview

Almost every JSP page in the application project has an Action class associated to them. Every Action class extends org.apache.struts.action.Action class and over-writes the execute() method. Each Action class also has an additional getActionForward() method. The execute method is the method the JSP calls, to perform the actions the page requires. The Action class controls the flow of the application and it uses the execute method to delegate the duties to other classes. The Action classes delegate most of the duties to the ManagementBean classes. The getActionForward method gets and returns the next JSP page or Action class to be forwarded to. All Action classes also have common ServletContext sc, MiscUtilBean mub, and MysqlDataSource ds attributes.

<table>
<thead>
<tr>
<th>Action</th>
<th>Methods</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>execute()</td>
<td>ServletContext sc;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MiscUtilBean mub;</td>
</tr>
<tr>
<td></td>
<td>getActionForward()</td>
<td>MysqlDataSource ds;</td>
</tr>
</tbody>
</table>

ActionForm classes are used in both the JSP pages and Action classes. ActionForm classes have attributes that have associated setter and getter methods that can be accessed from both the JSP pages and Action classes.

ManagementBean classes are used by the Action classes to make connection to the database and execute Sql queries on the database. All ManagementBean classes have 8 common methods and two common attributes. An interface IManagementBean was developed with theses 8 methods and a ManagementBean Class was developed that implements IManagementBean Interface. All of the ManagementBean classes extend the ManagementBean Class.

ValidationBean classes are used by the administration Actions classes to validate information gather from the JSP pages. All ValidationBean classes have 3 common methods and two common attributes. An IValidationBean Interface was developed with theses 3 methods and a ValidationBean Class was developed that implements IValidationBean Interface. All of the ValidationBean classes extend the ValidationBean Class. Figure 12.1 is a class diagram of IManagementBean, ManagementBean, ValidationBean, and ValidationBean.
12.2 Administration Classes

The CategoryBean Class is a container class designed to store category object information in its attributes and is used to be passed to other objects and JSP pages in the application.

CategoryValidationBean Class was designed to validate all the CategoryBean Class attributes.

The CategoryManagementBean Class was designed to make database connections and execute SQL queries to retrieve and update category information in the database.

The AdminCategoryAction Class is a controller class designed to retrieve the value set in the JSP page field variables, and perform different tasks based on the action field variable retrieved from the JSP page.

When the user clicks on any of the buttons on the categoryAdmin.jsp and categoryMaintenance.jsp pages, a hidden action variable is set on the page and the page is forwarded to the AdminCategoryAction Class. The AdminCategoryAction Class populates the CategoryBean Class attributes with the values retrieved from the JSP page variables and calls the CategoryValidationBean Class to validate the CategoryBean Class attributes. If the validation fails, the AdminCategoryAction Class redirects the user back to the JSP Page with an
error message. The AdminCategoryAction Class calls CategoryManagementBean Class to make database connections and execute queries on the database based on the CategoryBean Class attributes.

The CrustAdminBean Class is a container class designed to store crust object information in its attributes and is used to be passed to other objects and JSP pages in the application.

CrustValidationBean Class was designed to validate all the CrustAdminBean Class attributes.

The CrustManagementBean Class was designed to make database connections and execute Sql queries to get, insert, update and delete crust information in the database.

The AdminCrustAction Class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks based on the action field variable retrieved from the JSP page.

When the user clicks on any of the buttons on the crustAdmin.jsp and crustMaintenance.jsp pages, a hidden action variable is set on the page and the page is forwarded to the AdminCrustAction Class. The AdminCrustAction Class populates the CrustAdminBean Class attributes with the values retrieved from the JSP page field variables, and calls the CrustValidationBean Class to validate the CrustAdminBean Class attributes. If the validation fails, the AdminCrustAction Class forwards the user back to the JSP Page with an error message. The AdminCrustAction Class calls the CrustManagementBean Class to make database connections and execute queries on the database based on the CrustAdminBean Class attributes. Figure 12.2 is a class diagram for the crust and category admin classes.

The EmployeeAdminBean Class is a container class designed to store employee object information in its attributes and is used to be passed to other objects and JSP pages in the application.

The EmployeeValidationBean Class was designed to validate all the EmployeeAdminBean Class attributes.

The EmployeeManagementBean Class was designed to make database connections and execute Sql queries to get, insert, update, and delete employee information in the database.

The AdminEmployeeAction Class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks based on the action field variable retrieved from the JSP page.

When the user clicks on any of the buttons on the employeeAdmin.jsp page and employeeMaintenance.jsp page, a hidden action variable is set on the page and the page is forwarded to the AdminEmployeeAction Class. The AdminEmployeeAction Class populates the EmployeeAdminBean Class attributes with the values retrieved from the JSP page field
variables, and calls the EmployeeValidationBean Class to validate the EmployeeAdminBean Class attributes. If the validation fails, the AdminEmployeeAction Class forwards the user back to the JSP Page with an error message. The AdminEmployeeAction Class calls the EmployeeManagementBean Class to make database connections and execute queries on the database based the EmployeeAdminBean Class attributes.

The UserAdminBean Class is a container class designed to store user object information in its attributes, and is used to be passed to other objects and JSP pages in the application.

The UserValidationBean Class was designed to validate all the UserAdminBean Class attributes.

The UserManagementBean Class was designed to make database connections and execute Sql queries to get, insert, update and delete user information in the database.

The AdminUserAction Class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks based on the action field variable retrieved from the JSP page.
When the user clicks on any of the buttons on userAdmin.jsp and userMaintenance.jsp pages, a hidden action variable is set on the page and the user is forwarded to the AdminUserAction Class. The AdminUserAction Class populates the UserAdminBean Class attributes with the values retrieved from the JSP page field variables, and calls the UserValidationBean Class to validate the UserAdminBean Class attributes. If the validation fails, the AdminUserAction Class forwards the user back to the JSP Page with an error message.

The AdminUserAction Class calls the UserManagementBean Class to make database connections and execute queries on the database based on UserAdminBean Class attributes.

Figure 12.3 is a class diagram for the employee and user admin classes.

Figure 12.3: Class diagram for the employee and user admin classes.

The AdminSizeBean Class is a container class designed to store size object information in its attributes and is used to be passed to other objects and JSP pages in the application.

The SizeValidationBean Class was designed to validate all the AdminSizeBean Class attributes.

The SizeManagementBean Class was designed to make database connections and execute...
Sql queries to get, insert, update and delete size information in the database.

The AdminSizeAction Class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks based on the action field variable retrieved from the JSP page.

When the user clicks on any of the buttons on the sizeAdmin.jsp and sizeMaintenance.jsp pages, a hidden action variable is set on the page and the user is forwarded to the AdminSizeAction Class. The AdminSizeAction Class populates the AdminSizeBean Class attributes with the values retrieved from the JSP page field variables, and calls the SizeValidationBean Class to validate the AdminSizeBean Class attributes. If the validation fails, the AdminSizeAction Class forwards the user back to the JSP Page with an error message. The AdminSizeAction Class calls the SizeManagementBean Class to make database connections and execute queries on the database based on the AdminSizeBean Class attributes.

The AdminProdSizeBean Class is a container class designed to store size object information in its attributes and is used to be passed to other objects and JSP pages in the application.

The AdminProdSizeAction is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks based on the action field variable retrieved from the JSP page.

When the user clicks on any of the buttons on the sizeProdAdmin.jsp and sizeProdMaintenance.jsp pages, a hidden action variable is set on the page and the user is forwarded to the AdminProdSizeAction Class. The AdminProdSizeAction Class populates the AdminProdSizeBean Class attributes with the values retrieved from the JSP page field variables, and calls the SizeValidationBean to validate the AdminProdSizeBean Class attributes. If the validation fails the AdminProdSizeAction Class forwards the user back to the JSP Page with an error message. The AdminProdSizeAction Class calls the SizeManagementBean Class to make database connection and execute queries on the database based on the AdminProdSizeBean Class attributes. Figure 12.4 is a class diagram for the size and product size admin classes.

The ItemAdminBean Class is a container class designed to store product item object information in its attributes and is used to be passed to other objects and JSP pages in the application.

The ItemValidationBean Class was designed to validate all the ItemAdminBean Class attributes.

The ItemManagementBean Class was designed to make database connection and execute Sql queries to get, insert, update and delete product item information in the database. The AdminItemAction Class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks based on the action field variable re-
When the user clicks on any of the buttons on itemAdmin.jsp and itemMaintenance.jsp pages, a hidden action variable is set on the page and the user is forwarded to the AdminItemAction Class. The AdminItemAction Class populates the ItemAdminBean Class attributes with the values retrieved from the JSP page field variables, and calls the ItemValidationBean Class to validate the ItemAdminBean Class attributes. If the validation fails the AdminItemAction Class forwards the user back to the JSP Page with an error message. The AdminItemAction Class calls the ItemManagementBean Class to make database connection and execute queries on the database based on the ItemAdminBean Class attributes.

The ProductAdminBean Class is a container class designed to store product object information in its attribute and is used to be passed to other objects and JSP pages in the application.

The ProductValidationBean Class was designed to validate all the ProductAdminBean Class attributes.

The ProductManagementBean Class was designed to make database connection and ex-
execute SQL queries to get, insert, update and delete product information in the database.

The AdminProductAction Class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks based on the action field variable retrieved from the JSP page.

When the user clicks on any of the buttons productAdmin.jsp and productMaintenance.jsp pages, a hidden action variable is set on the page and the user is forwarded to the AdminProductAction Class. The AdminProductAction Class populates the ProductAdminBean Class attributes with the values retrieved from the JSP page field variables, and calls the ProductValidationBean Class to validate the ProductAdminBean Class attributes. If the validation fails the AdminProductAction Class forwards the user back to the JSP Page with an error message. The AdminProductAction Class calls the ProductManagementBean Class to make database connection and execute queries on the database based on the ProductAdminBean Class attributes. Figure 12.5 is a class diagram for the product and product item admin classes.

![Class diagram for the product and product item admin classes.](image)

Figure 12.5: Class diagram for the product and product item admin classes.

The InventoryAdminBean Class is a container class designed to store inventory object in-
formation in its attributes and is used to be passed to other objects and JSP pages in the application.

The InventoryValidationBean Class was designed to validate all the InventoryAdminBean Class attributes.

The InventoryManagementBean Class was designed to make database connection and execute SQL queries to get, insert, update and delete inventory information in the database.

The AdminInventoryAction Class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks based on the action field variable retrieved from the JSP page.

When the user clicks on any of the buttons inventoryAdmin.jsp and inventoryMaintenance.jsp pages, a hidden action variable is set on the page and the page is forwarded to the AdminInventoryAction Class. The AdminInventoryAction Class populates the InventoryAdminBean Class attributes with the values retrieved from the JSP page field variables, and calls the InventoryValidationBean Class to validate the InventoryAdminBean Class attributes. If the validation fails the AdminInventoryAction Class forwards the user back to the JSP Page with an error message. The AdminInventoryAction Class calls the InventoryManagementBean Class to make database connection and execute queries on the database based on the InventoryAdminBean attributes.

The InventProdAdminBean Class is a container class designed to store inventory product object information in its attributes and is used to be passed to other objects and JSP pages in the application.

The InventProdValidationBean Class was designed to validate all the InventProdAdminBean Class attributes.

The InventProdManagementBean Class was designed to make database connection and execute SQL queries to get, insert, update and delete inventory product information in the database.

The AdminInventProdAction Class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks based on the action field variable retrieved from the JSP page.

When the user clicks on any of the buttons productInventoryAdmin.jsp and inventProdMaintenance.jsp pages, a hidden action variable is set on the page and the user is forwarded to the AdminInventProdAction Class. The AdminInventProdAction Class populates the InventProdAdminBean Class attributes with the values retrieved from the JSP page variable and calls the InventProdValidationBean Class to validate the InventProdAdminBean Class attributes. If the validation fails the AdminInventProdAction Class forwards the user back to the JSP Page with an error message. The AdminInventProdAction Class calls the In-
ventProdManagementBean Class to make database connection and execute queries on the database based on the InventProdAdminBean Class attributes. Figure 12.6 is a class diagram for the inventory and inventory product admin classes.

12.3 ClockIn/ClockOut and Logon Classes

The UserBean Class is a container class designed to store the user information in its attributes, and is used by the application to determine what functionality the user has access to.

The LogonAction Class is a controller class designed to retrieve username and password field variables from the logon.jsp page.

When the user clicks on any of the buttons on the logon.jsp page, a hidden action variable is set on the page and the page is forwarded to the LogonAction Class. The LogonAction Class calls UserManagementBean Class to validate the user logon values and build a UserBean Class. If the user logon values are invalid, the user is forwarded to the logon.jsp page with an error message. When LogonAction Class retrieves the UserBean Class from

Figure 12.6: Class diagram for the inventory and inventory product admin classes.
the UserManagementBean Class, the LogonAction Class set the UserBean Class into the
memory to be used by other parts of the application.

The ClockAction Class is a controller class designed to retrieve username and password variables from the clockDisplay.jsp page.

When the user clicks on any of the buttons on the clockDisplay.jsp page, a hidden action variable is set on the page and the user page is forwarded to the ClockAction Class. The ClockAction Class uses the UserManagementBean to clock the user in or out of the application.

The LogoutAction Class is a controller class designed to remove the UserBean Class from the memory and closes the session opened by the user. When the user clicks the logout link from any page, the user is forwarded to the LogoutAction Class.

The NavAction Class is a controller class designed to navigate from the hyperlink pages to other parts of the application. When the user clicks on a hyperlink from any page, the user is forwarded to the NavAction Class. The NavAction Class uses the UserBean Class to validate the user access rights to the JSP page the user is trying access.

The AdminNavAction Class is a controller class designed to navigate from the hyperlink pages to other parts of the application. When the user clicks on a hyperlink from any of the Administrator pages, the user is forwarded to the AdminNavAction Class. The Administrator Functionality is a big module and therefore it was designed to have its own navigation class. The AdminNavAction Class uses the UserBean Class to validate the user access rights to the JSP page the user is trying access. Figure 12.7 is a class diagram for the clock, logon, logoff, and nav classes.

12.4 Order Classes

The CustomerManagementBean Class was designed to make database connections and execute Sql queries to get, insert, and update customer information in the database.

The OrderBean Class is a container class designed to store order information in its attributes, and is used to be passed to other objects and JSP pages in the application. The OrderBean Class contains a HashMap that stores multiple OrdertItemBeans that are added to the OrderBean Class.

The OrdertItemBean Class is a container class designed to store order item information in its attributes and is used to be passed to other objects and JSP pages in the application.

The OrderAction Class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks based on the action field variable retrieve for the JSP page.
The OrderItemAction Class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks based on the action field variable retrieve for the JSP page.

The process of taking and changing orders in the application is a complex process and involves many JSP pages. For all the JSP pages that are associated with the order process, there are multitudes of buttons the user can click on. It would be difficult to keep track of different actions that one OrderAction class would need to perform for every button clicked on by the user, therefore the order process is divided into multiple OrderAction classes.

The user moves through the orderEntry.jsp page and orderAddress.jsp page to gather information about the order. When the user clicks on any button on the orderEntry.jsp page and orderAddress.jsp page, the user is forwarded to the OrderAction Class. The OrderAction Class calls the CustomerManagementBean Class to retrieve or insert the customer information. The OrderAction Class also creates an OrderBean Object and sets the OrderBean attributes retrieved from each JSP page field variables. The OrderActions Class places the OrderBean in memory and forwards the user the next JSP page.
The user traverses through the orderItem.jsp page, orderItemsTypes.jsp and orderItemsTypep2.jsp pages to gather information on an order item to add to the order. When the user clicks on any button on these pages, the user is forwarded to the OrderItemAction Class. The OrderItemAction Class retrieves the JSP page field variables and stores the values in the OrderItemBean Class attributes. The OrderItemAction Class calls the ItemManagementBean Class to retrieve a list of product items. The OrderItemBean Class is added to the OrderBean Class before the OrderItemAction Class forwards the user to the orderSum.jsp page.

The OrderItemChangeAction Class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks base on the action field variable retrieve for the JSP page.

When the user wants to make changes to an order item, the orderItemsChangeTypes.jsp and orderItemsChangeTypep2.jsp pages are used to make the changes. When the user clicks on any button on these pages, the user is forwarded to the OrderItemChangeAction Class. The OrderItemChangeAction Class retrieves the JSP page field variables and updates the values in the OrderItemBean Class attributes. The OrderItemChangeAction Class calls the ItemManagementBean Class to retrieve a list of product items. The OrderItemBean Class is updated to the OrderBean Class before the OrderItemAction Class forwards the user to the orderSum.jsp page. Figure 12.8 is a class diagram for classes associated with order, order and order change actions.

The OrderManagementBean Class was designed to make database connection and execute Sql queries to get, insert, and update information about an order, order items and order extras in the database.

The OrderSumAction Class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks base on the action field variable retrieve for the JSP page.

When the user clicks on any button on the orderSum.jsp page, the user is forwarded to the OrderSumAction Class. The OrderSumAction Class is used to make the final submission of an order and is also used to forward the user to other order JSP pages to add new, make changes to, or delete an order item. The OrderSumAction Class retrieves the OrderBean from the memory and calls the OrderManagementBean Class to either update or create a new order in the database.

The OrderEditAction Class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks base on the action field variable retrieve for the JSP page.

When the user clicks on any button on the orderEdit.jsp page, the user is forwarded to the OrderEditAction Class. The OrderEditAction Class retrieves the selected order from the orderEdit.jsp page and calls the OrderManagementBean Class to build an OrderBean
class. The OrderEditAction class places the OrderBean in the memory and forwards the user to the orderSum.jsp page. Figure 12.9 is a class diagram for classes associated with order sum and order edit actions.

12.5 Dispatch/Cashier Classes

The CashoutOrderForm class is a subclass of ActionForm and is designed to be passed within the execute method of the CashoutOrderAction class. It is also integrated into the cashoutOrder.jsp page field variables by setting each JSP tag properties values to each of the CashoutOrderForm class attribute names. This makes it easier for getting and setting values on the JSP page, without making calls to the JSP page field variables directly.

The CashierAction class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks based on the action field variable retrieved from the JSP page.

The CashoutOrderAction class is a controller class designed to retrieve the information set in the CashoutOrderForm class attributes, and perform different tasks based on the
CashoutOrderForm action attribute value.

The CashierAction Class is used in the first step of cashing out the order process. The CashierAction first calls the OrderManagementBean Class to build an OrderBean Class list and forwards the user to the cashierMain.jsp page. When the user clicks on any button on the cashierMain.jsp page, the user is forwarded to the CashierAction Class. The main purpose of the CashierAction Class is to retrieve the selected order from cashierMain.jsp page, pull the OrderBean Class for the selected order form the OrderBean Class list, and forward it to the CashoutOrderAction Class.

The CashoutOrderAction Class is used to cash out a dine-in or carry-out order. The CashoutOrderAction Class retrieves OrderBean Class from the memory and populates the attributes of CashoutOrderForm with the OrderBean Class. The CashoutOrderAction forwards the user to cashoutOrder.jsp page. When the user makes changes to the cashoutOrder.jsp, it is made automatically in the CashoutOrderForm. When the user clicks on the close button on the cashoutOrder.jsp page, the user is forward to the CashoutOrderAction Class. The CashoutOrderAction Class calls the OrderManagementBean Class to update the order status to closed. The CashoutOrderAction Class then forwards the user
back to the cashierMain.jsp page. Figure 12.10 is a class diagram for classes associated with cashier and cash out actions.

The DriverManagementBean was designed to make database connections and execute Sql queries to get, insert, and update driver information in the database.

The DispatcherAction Class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks based on the action field variable retrieved from the JSP page.

The DispatchOrderAction Class is a controller class designed to retrieve the information set in the DispatchOrderForm Class attributes, and performs different tasks based on the DispatchOrderForm action attribute value.

The DispatchOrderForm Class is subclass of ActionForm and is designed to be passed in the execute method of the DispatchOrderAction Class. It is also integrated in the dispatchOrder.jsp page field variables by setting each JSP tag properties values to each of the DispatchOrderForm Class attribute names. This makes it easier getting and setting values.
on the JSP page without making call to the JSP page field variables directly.

The DispatchOrderListBean was designed for controlling a series of order lists to display on the dispatchOrder.jsp page. A driver can take up to five orders on a delivery, but there may not be five orders to deliver, so the dispatchOrder.jsp page needs a way of displaying a series of one to five order lists and associate each list to an order attribute in the DispatchOrderForm. The DispatchOrderListBean provides a way of displaying a series of order lists on the dispatchOrder.jsp and associates each list to an order attribute in the DispatchOrderForm.

When the user clicks on any button on the dispatch.jsp page, the user is forwarded to the DispatcherAction Class. The DispatcherAction Class calls the DriverManagementBean Class to build an active driver list and calls OrderManagementBean Class to build a delivery order list. After the lists are built, the DispatcherAction Class forwards the user to the dispatch.jsp. When the user clicks on any button on the dispatch.jsp page, the user is forwarded to the DispatcherAction Class. The DispatcherAction Class retrieves the selected orders and the selected driver from the dispatch.jsp page and passes the values to the DispatchOrderAction Class.

The DispatchOrderAction Class is used to dispatch the orders to a driver. The DispatchOrderAction Class retrieves the selected order and driver values from the memory, and sets the orders and driverid attributes in the DispatchOrderForm Class. The DispatchOrderAction Class calls the DriverManagementBean Class to build a driver list and sets the driver list attribute in the DispatchOrderForm Class. The DispatchOrderAction Class calls the OrderManagementBean Class to build an order list and sets the order list attribute in the DispatchOrderForm Class. The DispatchOrderAction Class builds a DispatchOrderListBean list based on the order list and sets the DispatchOrderListBean list attribute in the DispatchOrderForm Class. The DispatchOrderAction Class then forwards the user to the dispatchOrder.jsp.

When the user clicks on the submit button on the dispatchOrder.jsp page, the action attribute is set in DispatchOrderForm Class and the user is forwarded to the DispatchOrderAction Class. The DispatchOrderAction Class retrieves the driverid and the orders from the DispatchOrderForm attributes. The DispatchOrderAction Class calls the OrderManagementBean Class to update each order status to "dispatch" and the driver is set to the driverid. The DispatchOrderAction Class calls the DriverManagementBean Class to set the driver status to out. Figure 12.11 is a class diagram for classes associated with dispatch actions.

The DriverBankForm Class is subclass of ActionForm and is designed to be passed in the execute method of the DriverBankAction Class. It is also integrated in the driverBank.jsp page field variables by setting each JSP tag properties values to each of the DriverBankForm Class attribute names. This makes it easier getting and setting values on the JSP page without making call to the JSP page field variables directly.
Figure 12.11: Class diagram for the classes associated with dispatch actions.

The DriverBankAction Class is a controller class designed to retrieve the attributes values from the DriverBankForm Class, and performs different tasks based on the DriverBankForm Class action attribute value.

When the user clicks on any of the buttons on driverBank.jsp page, the action attribute is set in DriverBankForm Class and the user is forwarded to the DriverBankAction Class. The DriverBankAction Class calls the DriverManagementBean class to create a driver in the database based on the attributes from the DriverBankForm Class. The DriverBankAction forwards the user to the dispatchMain.jsp page.

The DriverInForm Class is subclass of ActionForm and is used to be passed in the execute method of the DriverInAction Class. It is also integrated in the driverIn.jsp page field variables by setting each JSP tag properties values to each of the DriverInForm Class attribute names. This makes it easier getting and setting values on the JSP page without making call to the JSP page field variables directly.

The DriverInAction Class is a controller class designed to retrieve the attributes values from the DriverInForm Class, and performs different tasks based on the DriverInForm Class ac-
tion attribute value.

When the user is forwarded to the DriverInAction Class from the SelectDriverInAction, the DriverInAction Class retrieves the driverid from memory and sets the driverid attribute in DriverInForm Class. The DriverInAction Class calls the OrderManagementBean Class to build an order list that has been assigned to the driver and sets the orderList attribute in the DriverInForm Class. The DriverInAction Class then forwards the user to the driverIn.jsp page.

When the user clicks on any button on the driverIn.jsp page, the action attribute is set in DriverInForm Class and the user is forwarded to the DriverInAction Class. When all the orders are cashed out the DriverInAction Class calls the DriverManagementBean Class to sets the driver status to in, and forwards the user to the dispatchMain.jsp page. Figure 12.12 is a class diagram for classes associated with driver in and driver bank actions.

The SelectDriverForm Class is subclass of ActionForm and is designed to be passed in the execute method of the SelectDriverAction Class, SelectDriverInAction. Class, and
UnassignOrdersAction Class. It is also integrated in the selectDriver.jsp, selectDriverIn.jsp and unassignOrders.jsp pages field variables by setting each JSP tag properties values to each of the SelectDriveForm Class attribute names. This makes it easier getting and setting values on the JSP page without making call to the JSP page field variables directly.

The SelectDriverAction Class is a controller class designed to retrieve the attributes values from the SelectDriverForm Class, and performs different tasks based on the SelectDriverForm Class action attribute value.

When the user is forwarded to SelectDriverAction form the dispatchMain.jsp page, the SelectDriverAction Class calls the DriverManagementBean Class to build an active drive list. The SelectDriverAction Class sets the driverlist attribute in the SelectDriverForm class with drive list and forwards the user to the selectDriver.jsp page.

When the user clicks on any of the buttons on the selectDriver.jsp page, the action attribute is set in SelectDriverForm Class and the user is forwarded to the SelectDriverAction Class. The SelectDriverAction Class gets the driverid attribute from the SelectDriverForm class and places it in the memory, and forwards user to the CashoutDriverAction Class.

The SelectDriverInAction Class is a controller class is designed to retrieve the attributes values from the SelectDriverForm Class and performs different tasks based on the SelectDriverForm Class action attribute value.

When the user is forwarded to SelectDriverInAction form the dispatchMain.jsp page, the SelectDriverInAction Class calls the DriverManagementBean Class to build an active drive list. The SelectDriverInAction Class sets the driverlist attribute in the SelectDriverForm class with drive list and forwards the user to the selectDriverIn.jsp page.

When the user clicks on any of the buttons on the selectDriverIn.jsp page, the action attribute is set in SelectDriverForm Class and the user is forwarded to the SelectDriverInAction Class. The SelectDriverInAction Class gets the driverid attribute from the SelectDriverForm class and places it in the memory, and forwards user to the DriverInAction Class.

The UnassignOrdersAction Class is a controller class designed to get the attribute values form the SelectDriverForm Class, and performs different tasks based on the SelectDriverForm Class action attribute value.

When the user is forwarded to UnassignOrderAction form the dispatchMain.jsp page, the UnassignOrderAction Class calls the DriverManagementBean Class to build an active drive list. The UnassignOrderAction Class sets the driverlist attribute in the SelectDriverForm class with drive list and forwards the user to the unassignOrder.jsp page.

When the user clicks on any of the buttons on the unassignOrder.jsp page, the action attribute is set in SelectDriverForm Class and the user is forwarded to the UnassignOrderAction Class. The UnassignOrderAction Class gets the driverid calls OrderManagementBean
and sets the status to "open" for the orders dispatch to the driver. The UnassignOrderAction Class calls the DriveManagementBean and sets the drive status to in. The UnassignOrder-Action forwards the user to the dispatchMain.jsp page.

The CashoutDriverForm Class is subclass of ActionForm and is designed to be passed in the execute method of the CashoutDriverAction Class. It is also integrated in the cashoutDriver.jsp page field variables by setting each JSP tag properties values to each of the CashoutDriverForm Class attribute names. This makes it easier getting and setting values on the JSP page without making call to the JSP page field variables directly.

The CashoutDriverAction Class is a controller class designed to get the attribute values from the CashoutDriverForm Class, and perform different tasks base on the CashoutDriverForm Class action attribute value.

When the user is forwarded to CashoutDriverAction Class is from the SelectDriverIn-Action, the CashoutDriverAction Class retrieves the driverid from memory and sets the driverid attribute in CashoutDriverForm Class. The CashoutDriverAction Class calls the OrderManagergmentBean Class to get the totals for orders delivered by the drive. The CashoutDriverAction Class uses the totals to set other attributes in the CashoutDriverForm Class. The CashoutDriverAction Class then forwards the user to the cashoutDriver.jsp page.

When the user clicks on any of the buttons on cashoutDriver.jsp page, the action attribute is set in CashoutDriverForm Class and the user is forwarded to the CashoutDriverAction Class. When driver owes no money the CashoutDriverAction Class calls the DriverManagementBean Class set the drive to inactive. The DriverInAction Class forwards the user to the dispatchMain.jsp page. Figure 12.13 is a class diagram for classes associated with cash out driver, select driver and unassigned order actions.

12.6 Management Classes

The InventoryDeliveryBean Class is a container class designed to store delivered inventory item information in its attributes and is used to be passed to other objects and JSP pages in the application.

The DeliveryInventoryAction Class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks based on the action field variable retrieved from the JSP page.

The EditInventoryAction Class is a controller class designed to retrieve the information set in the JSP page field variables, and perform different tasks based on the action field variable retrieve for the JSP page.

When the user clicks on any of the buttons on the inventoryDelivery.jsp and inventoryDeliveryItem.jsp page, a hidden action variable is set on the page and the user is forwarded
Figure 12.13: Class diagram for the classes associated with cash out driver, select driver and unassigned actions.

to the DeliveryInventoryAction Class. The DeliveryInventoryAction Class populates the InventoryDeliveryBean Class attributes with the values retrieve from the inventoryDeliveryItem.jsp field variables and add InventoryDeliveryBean Class to a delivery list and forwards the list to inventDelivery.jsp page.

When the user is done adding the InventoryDeliveryBeans to the delivery list and clicks on the submit button, the user is forwarded to the DeliveryInventoryAction Class. The DeliveryInventoryAction Class calls the InventoryManagementBean to update the database withs the inventory items delivered.

When the user clicks on any of the buttons on the inventManagement.jsp and inventoryEdit.jsp page, a hidden action variable is set on the page and the user is forwarded to the EditInventoryAction Class. The EditInventoryAction first gets the select inventory item from inventManagement.jsp, populate a InventoryAdminBean Class, and places the InventoryAdminBean Class into memory. The EditInventoryAction then forwards the user to the inventoryEdit.jsp page.
When the user finished making changes to the inventory amount, and clicks on the submit button, the user is forwarded to the EditInventoryAction Class. The EditInventoryAction retrieves values from the inventoryEdit.jsp page field variables and the calls InventoryManagementBean to update the database with the changes to amount for the inventory items on hand. The EditInventoryAction Class then forwards the user to inventManagement.jsp page. Figure 12.14 is a class diagram for classes associated with inventory edit and inventory delivery actions.

Figure 12.14: Class diagram for the classes associated with inventory edit and inventory delivery actions.

The EmployeeScheduleBean is a container class designed to store employee weekly schedule information in its attributes and is used to be passed to other objects and JSP pages in the application.

The ScheduleForm Class is subclass of ActionForm and is designed to be passed in the execute methods of EmployeScheduleAction and ScheduleActionDisplayAction Classes. The ScheduleForm is integeted into scheduleMain.jsp and empSchedule.jsp pages It is also integeted in the JSP page field variables by setting each JSP tag properties values to each of the ScheduleForm Class attribute names. This makes it easier getting and setting
values on the JSP page without making call to the JSP page field variables directly.

The ScheduleManagementBean Class was designed to make database connections and execute SQL queries to get, insert, and update employee schedule information in the database.

The ScheduleAction Class is a controller class designed to get the attribute values from the ScheduleForm Class, and performs different tasks based on the ScheduleForm Class action attribute value.

The EmployeeScheduleAction Class is a controller class designed to get the attribute values from the ScheduleForm Class, and performs different tasks based on the ScheduleForm Class action attribute value.

When the user is forwarded to ScheduleAction Class from manageMain.jsp, the ScheduleAction Class calls ScheduleManagementBean Class to generate a list of EmployeeScheduleBean Class. After the list is generated, the ScheduleAction Class sets the ScheduleForm Class list attribute and forwards the user to the scheduleMain.jsp page. When the user clicks on a button to edit a select employee schedule, the scheduleMain.jsp forwards the user to the EmployeeScheduleAction Class.

When changes are made to the EmployeeScheduleBean Class the EmployeeScheduleAction Class updates ScheduleForm Class list attribute with the changes in the EmployeeScheduleBean Class. The EmployeeScheduleAction Class then forwards the user to the EmployeeScheduleAction Class.

When the user clicks the save button on the scheduleMain.jsp page the user is forwarded to the EmployeeScheduleAction Class. The EmployeeScheduleAction Class calls ScheduleManagementBean Class to save the changes made to the schedule and forwards the user to the manageMain.jsp. Figure 12.15 is a class diagram for classes associated with schedule edit and employee schedule actions.


The ReportManagementBean Class was designed to make database connections and generate execute SQL queries to get information from the database to generate a list of report
beans for various reports.

The EmployeeReportBean is a container designed that is used to stored employee report display data in its attributes and is used to be passed to other objects and JSP pages in the application.

The EmployeeReportStartAction Class is a controller class designed to get the attribute values from the ReportDateForm Class, and performs different tasks based onto the ReportDateForm Class action attribute value.

The EmployeeReportDisplayAction Class is a controller class used to get the attribute values from the ReportDateForm Class, and performs different tasks based on the ReportDate-Form Class action attribute value.

When the user clicks on any of the buttons on the employeeReportStart.jsp page, the action attribute is set in the ReportDateForm Class and the user is forwarded to the EmployeeReportStartAction Class. The EmployeeReportStartAction Class then forwards the user to the
EmployeeReportDisplayAction Class.

The EmployeeReportDisplayAction Class is used to retrieve values from the ReportDateForm Class attributes and calls the ReportManagementBean Class to generate a report list of EmployeeReportBean Classes. After the report list is generated the EmployeeReportDisplayAction Class sets the ReportDateForm list attribute and forwards the user to the employeeReportDisplay.jsp page. When the user clicks on a button on the employeeReportDisplay.jsp page, the EmployeeReportDisplayAction Class forwards the user to reportMain.jsp page.

The DriverReportBean is a container class designed to store driver report display data in its attributes and is used to be passed to other objects and JSP pages in the application.

The DriverReportStartAction Class is a controller class used to get the attribute values from the ReportDateForm Class, and performs different tasks based on the ReportDateForm Class action attribute value.

The DriverReportDisplayAction Class is a controller class used to get the attribute values from the ReportDateForm Class, and performs different tasks based on the ReportDateForm Class action attribute value.

When the user clicks on any of the buttons on driverReportStart.jsp page, the action attribute is set in ReportDateForm Class and the user is forwarded to the DriverReportStartAction Class. The DriverReportStartAction Class then forwards the user on to the DriverReportDisplayAction Class.

The DriverReportDisplayAction Class is used to retrieve the values from the ReportDateForm Class attributes and calls the ReportManagementBean Class to generate a report list of DriverReportBean Classes. After the report list is generated the DriverReportDisplayAction Class forwards the user to the driverReportDisplay.jsp page. When the user clicks on a button on the driverReportDisplay.jsp page the DriverReportDisplayAction Class forwards the user to reportMain.jsp page. Figure 12.16 is a class diagram for classes associated with driver report and employee report actions.

The OrderItemReportBean is a container class designed to store order item report display data in its attributes and is used to be passed to other objects and JSP pages in the application.

The OrderItemReportStartAction Class is a controller class designed to get the attribute values from the ReportDateForm Class, and performs different tasks based on the ReportDateForm Class action attribute value.

The OrderItemReportDisplayAction Class is a controller class designed to get the attribute values from the ReportDateForm Class, and performs different tasks based on the ReportDateForm Class action attribute value.
When the user clicks on any of the buttons on productReportStart.jsp page, the action attribute is set in ReportDateForm Class and the user is forwarded to the OrderItemReportStartAction Class. The OrderItemReportStartAction Class then forwards the user on to the OrderItemReportDisplayAction Class.

The OrderItemReportDisplayAction Class is used to retrieve the values from the ReportDateForm Class and call the ReportManagementBean Class to generate a report list of OrderItemReportBean Classes. After the report list is generated the OrderItemReportDisplayAction Class forwards the user to the productReportDisplay.jsp page. When the user clicks on a button on the productReportDisplay.jsp page, the OrderItemReportDisplayAction Class forwards the user to reportMain.jsp page.

The InventoryReportBean is a container class designed to store inventory report display data in its attributes and is used to be passed to other objects and JSP pages in the application.
The `InventoryReportStartAction` class is a controller class designed to get the attribute values from the `ReportDateForm` class, and performs different tasks based on the `ReportDateForm` class action attribute value.

The `InventoryReportDisplayAction` is a controller class designed to get the attribute values from the `ReportDateForm` class, and performs different tasks based on the `ReportDateForm` class action attribute value.

When the user clicks on any of the buttons on `inventoryReportStart` page, the action attribute is set in `ReportDateForm` class and the user is forwarded to the `InventoryReportStartAction` class. The `InventoryReportStartAction` class then forwards the user onto the `InventoryReportDisplayAction` class.

The `InventoryReportDisplayAction` class is designed to retrieve the values from the `ReportDateForm` class and call the `ReportManagementBean` class to generate a report list of `InventoryReportBean` classes. After the report list is generated the `InventoryReportDisplayAction` class forwards the user to the `inventoryReportDisplay.jsp` page. When the user clicks on a button on the `inventoryReportDisplay.jsp` page, the `InventoryReportDisplayAction` class forwards the user to `reportMain.jsp` page. Figure 12.17 is a class diagram for classes associated with order item report and inventory report actions.

The `OrderTypeReportBean` is a container class designed to store order type report display data in its attributes and is used to be passed to other objects and JSP pages in the application.

The `OrderTypeReportStartAction` class is a controller class designed to get the attribute values from the `ReportDateForm` class, and perform different tasks based on the `ReportDateForm` class action attribute value.

The `OrderTypeReportDisplayAction` is a controller class designed to get the attribute values from the `ReportDateForm` class, and perform different tasks based on the `ReportDateForm` class action attribute value.

When the user clicks on any of the buttons on `orderTypeReportStart.jsp` page, the action attribute is set in `ReportDateForm` class and the user is forwarded to the `OrderTypeReportStartActionClass`. The `OrderTypeReportStartActionClass` then forwards the user to the `OrderTypeReportDisplayActionClass`.

The `OrderTypeReportDisplayActionClass` is used to retrieve the values from the `ReportDateForm` class and call the `ReportManagementBean` class to generate a report list of `OrderTypeReportBean`. After the report list is generated the `OrderTypeReportDisplayActionClass` forwards the user to the `orderTypeReportDisplay.jsp` page. When the user clicks on a button on the `orderTypeReportDisplay.jsp` page, the `OrderTypeReportDisplayActionClass` forwards the user to `reportMain.jsp` page. Figure 12.18 is a class diagram for classes associated with order type report actions.
12.7 Other Classes

The ResourceManagerListener Class implements ServletContextListener and is designed to initialize resources. When the application is started up, the ResourceManagerListener Class is used to initialize resources in the application and load them into the memory. One of the resources is a list of error messages. The ResourceManagerListener Class uses the MessageManagementBean Class to build a MessageBean Class. The MessageBean Class is created with a Map array of error messages. The ResourceManagerListener Class loads the MessageBean Class into the memory after the MessageBean Class.

ErrorMessageBean Class is designed to display and format an error message on a JSP page. The ErrorMessageBean Class uses the MessageBean Class to retrieve the error message to display.

The GetCrustTypes Class is a Servlet that is used by the orderItem.jsp to load a list of crust items for user to select from a drop down menu on the JSP page. The GetCrust-
Figure 12.18: Class diagram for the classes associated with order type report actions.

Types Class calls the ProductManagementBean to make a database connection and query the database for a list crust items.

The GetProductTypes Class is a Servlet that is used by the orderItem.jsp to load a list of products for user to select from a drop down menu on the JSP page. The GetProductTypes Class calls the ProductManagementBean to make a database connection and query the database for list products. Figure 12.19 is a class diagram of other classes.
Figure 12.19: Class diagram for other classes.
Chapter 13

Database

A database schema was created called RTWADB, it is short for Restaurant Web Application Database. There are 8 columns that every table has: DTCR, TMCR, APCR, USCR, DTLM, TMLM, APLM, and USLM columns.

- **DTCR** - date data type - Used to keep track of date created
- **TMCR** - time data type - Used to keep track of time created
- **APCR** - varchar(10) data type - Used to keep track of the part of the application that created it
- **USCR** - varchar (10) data type - Used to keep track of the user that created it
- **DTLM** - date data type - Used to keep track of date last maintained.
- **TMLM** - time data type - Used to keep track of time last maintained.
- **APLM** - varchar (10) data type - Used to keep track of the part of the application that last maintained it
- **USLM** - varchar (10) data type - Used to keep track of the user that last maintained it

A customer table was created called customers. The customers entity is based on the customer’s phone number.

An address table was created called addresses. The addresses table is used to store addresses of customers and is linked to the customer table by the customer phone number. Since this is the age of cellular phones, the customer has the ability to have multiple addresses per phone number and vice versa.

Three tables were created that are related to employees: the employee, empschedule, and clkusr. The employee table stores general information about the employee. The empschedule table stores information about the employee’s work schedule. The clkuser table stores information about when the employee clocks in and out. These tables are all linked based on employeeid from the employee table.

The wapusr table was created for users to logon to the web application. It stores user access rights to the application.

There are also two tables created for the drivers. The drusr table was created to keep track of available drivers to deliver orders. The commission table was also created to set a commission rate for the drivers.

An error table was created to store error messages called errmsg. The errmsg table is loaded into memory when the web application is started. Figure 13.1 is a database diagram of the users and employees tables.

The prodcrst, prodsize, prodtop, product, sizes, and category tables are created for prod-
ucts. They are associated with products that are sold in the restaurant. The category table is just a list of category types for the products. The size table is a list of size information. The product table stores general information about the products and is associated to a category. The prodsize is a table that associates the size of product based on the size and category. The prodcrst was created for pizza product crust and is associated to the product based on the product id and the size. The prodtop is a table created for any extra items that are associated with the product for instance pepperoni to the pizza. Figure 13.2 is a database diagram of the product tables.

The invdel, inventitems, inventprod, Inventcrust, Inventtop, and invonhnd tables were created for inventory. They are associated with inventory of products sold in the restaurant. The inventitems table store generals information of an inventory item. The invdel table was created to keep track of inventory items that are delivered to the restaurant. The invonhnd table was created to keep track of the inventory on hand in the restaurant. The inventprod table associates the inventory item with the products. The application uses the inventprod table to deduct amounts from the invonhnd table when an order is taken. The inventcrust table associates the inventory item with the products. The application uses the inventcrust table to deduct amounts from the invonhnd table when an order is taken. The inventtop table associates the inventory item with the products. The application uses the inventtop table to deduct amounts from the invonhnd table when an order is taken. By associating the inventory items to the product, product crust, and product topping, the application can deduct the amount of the inventory item from the invonhnd table based on the amount associated with the product, product crust, and product topping. Figure 13.3 is a database
Figure 13.2: Database diagram of the product tables.

diagram of the inventory tables.

The orders, ordextra and orditem tables were created to handle the order taking process of the application. The orders table keeps track of high level information about the order. It keeps track of the customer, address and credit card information associated with the order. The orditem table stores the product items that the customer selected for the order. The ordextra table is used to store the extras associated with the order item. That would include topping for a pizza or extra cheese for a sub sandwich. Figure 13.4 is a database diagram of the order tables.
Figure 13.3: Database diagram of the inventory tables.

Figure 13.4: Database diagram of the order tables.
14.1 Goals of Application

Before the start of the project, there were several goals set for development and implementation of the application.

1. Reduce the cost of development and implementation application.
   a. This means, If possible, using free open source license.

2. The server technology must adequate for the uses of the application.
   a. It must be able to perform well with up to 500 users.
   b. It must be scalable.
   c. It must have an ability to implement some type of security.

3. All the parts of the application must be developed as a web application, except for batch processes that are schedule to run throughout the day.
   a. A web application run at a centralize location and can be access by a web browser.
   b. No software installation needed on client computer.

4. The application must have an administration functionality.
   a. To maintain product objects in the application.
   b. To maintain inventory objects in the application.
   c. To maintain users in the application.
   d. To maintain employees in the application.

5. The application must have management functionality.
   a. To manage employee.
   b. To manage inventory
   c. To run reports.

6. The application must be able to take, change, and cancels orders.

7. The application must be able to cash out orders.

8. The application must be able to dispatch orders.

9. The application must be able to maintain drivers.

10. The application must have a way for a user to logon onto the application.

11. The application must have a way for an employee to clock in and out.
14.2 Reference Analysis

The references in this paper were used to explain concept and tools used in developing the application. The references are used to verify that the technology and tools used in developing the application can achieve the above goals.

1. Open source.
   a. Reference [2] “encyclopedia2. Open Source” gives a definition of open source software and open source licenses. This reference was used in this paper to justify the uses of open source technologies. The technologies used in development and implementation of this project utilize open source software with free open source licenses. This lowers the cost of implementation of the Pizza Store application.

2. Ubuntu Server - the references below are use to justify the use of Ubuntu Server for the Pizza Store application.
   a. Reference [1] “Ubuntu Documentation. About Ubuntu” is documentation on the benefits using Ubuntu. It proves that Ubuntu Server it is as powerful as other server software available in the market place. Ubuntu Server was chosen for the Pizza Store application because it provided a powerful server operating system at no cost to use.
   b. Reference [6] “Ubuntu Licensing. About Ubuntu” explain the free open source licenses agreement for the uses of Ubuntu Server. This benefits the development and implementation of the Pizza Store application.

3. MySQL - the references below are use to justify the use of MySQL Database Server for the Pizza Store application.
   b. Reference [14] “Dispelling the myths. MySQL AB” and [24] “Wikipedia. MySQL history” explains the features of MySQL. The references explain the free open source licenses agreement for the uses MySQL Database Server. With the information gather from these references, it shows that MySQL provides an adequate a database server for the application.


5. Model View Controller
a. Reference [23] “Wikipedia. Model-View-Controller” shows a software architecture concept behind the development of the application. It is also a concept software architecture concept behind the J2EE Platform and Struts Framework.

6. J2EE Platform


d. Reference [8] “Oracle. HttpServlet Interface” examines the API behind enterprise web application on a J2EE Platform. Apache Tomcat is an application server that provides an implementation of the J2EE and provides support for implementation of HttpServlet Interface.

e. Reference [9] “Oracle. HttpServletRequest Interface” examines the API behind enterprise web application on a J2EE Platform. Apache Tomcat is an application server that provides an implementation of the J2EE and provides support for implementation of HttpServletRequest Interface.

f. Reference [10] “Oracle. HttpServletResponse Interface” examines the API behind enterprise web application on a J2EE Platform. Apache Tomcat is an application server that provides an implementation of the J2EE and provides support for implementation of HttpServletResponse Interface.

g. Reference [11] “Oracle. HttpSession Interface” examines the API behind enterprise web application on a J2EE Platform. Apache Tomcat is an application server that provides an implementation of the J2EE and provides support for implementation of HttpSession Interface.

h. Reference [12] “Oracle. Servlet Interface” examines the API behind enterprise web application on a J2EE Platform. Apache Tomcat is an application server that provides an implementation of the J2EE and provides support for implementation of Servlet Interface.


7. Other Technologies

a. Reference [16] “w3schools. Javascript tutorial” shows the uses of Javascript in developing a web page. Javascript is utilizes in the development of a JSP.
b. Reference [18] “w3schools. XML tutorial” explains the uses of XML in developing web application and other application.


d. Reference [15] “w3schools. DTD tutorial” examines the use of DTD to define the XML structure with a list of legal elements and attributes. DTD is used by struts-config.xml in the Struts Framework to define the XML structure.

e. Reference [19] “Russ Weakley. Teach Yourself CSS. Sams Publishing” shows the uses of CSS in developing web pages. CSS is used to create reusable to common JSP page formats for table, column headers, column content and other aspects of the JSP page.

14.3 Market Analysis of Applications

As stated before, this project was designed to develop an application for a small to medium size restaurant business, in this particular, a pizza restaurant. At the time of development of this application, there were other older applications developed for the pizza restaurant business. In particular, Pizza Hut had an older application that was command line base. The application was developed to be in a Dos or Bash terminal. The user maneuvered through the application by pressing on the key associated to the action the user wanted to take. For instance, if the user wanted to place an order an “o” would be pressed to go to the order taking section. The whole application is a key based system. The reason for developing the application discussed in this paper was based on the Pizza Hut application. The application was developed to do everything the Pizza Hut application did, but developed using a newer web based technology.

As you can see below, The Pizza Hut application and the application that was developed for this paper (Pizza Store) has similar features. One of the big differences between the Pizza Hut and the Pizza Store applications is the Display feature. The Pizza Hut application display is a black screen with green text. The Pizza Store is a web base application, and was developed for a Windows based system and utilizes a web browser for display purpose. Another difference is the Navigation feature, the Pizza Hut system navigates through the application by pressing keys on the keyboard. The Pizza Store system navigates through the application by clicking on the hyperlinks with the mouse. This makes it easier and faster to move through the application because you don’t have to click on different buttons to navigate to different parts of the application. All the user has to do is click on the same mouse button. The next difference is the Logon as a role or user. The Pizza Hut application at the time, only had a role base logon, the application had one or two common logon users per role. The Pizza Store allows the user to create users and assign them roles. This is useful for keeping track of what is being done in the application and who is performing the tasks. It is also useful for reporting purposes. The last difference is the auto decrement of inventory when a product is sold. When this application was being develop the Pizza Hut application did not auto decrement the inventory when a product was sold. It relied on the
manager to adjust the inventory manually every day. The Pizza Store has an auto decrement of inventory feature when a product is sold in the store. This makes it easy for the manager to keep track of inventory and place accurate orders for inventory items. The manager still has the ability to adjust the inventory for shortages. The Pizza Hut application connects to the server through a dummy terminal and the process work load is fully placed on the server. The Pizza Store is a web application and the process is divided between the server and the web browser connected to the server. Figure 14.1 is a table of Pizza Hut Features and Pizza Store comparisions.

<table>
<thead>
<tr>
<th>Features</th>
<th>Pizza Hut</th>
<th>Pizza Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display</td>
<td>Green Screen</td>
<td>Browser</td>
</tr>
<tr>
<td>Navigation</td>
<td>Key base</td>
<td>Mouse click</td>
</tr>
<tr>
<td>Server Connection</td>
<td>Dumb terminal</td>
<td>Web server</td>
</tr>
<tr>
<td>Administrator process</td>
<td>?</td>
<td>yes</td>
</tr>
<tr>
<td>Clocking in and out process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Logon process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Logon as role user</td>
<td>as role only</td>
<td>as user with a role</td>
</tr>
<tr>
<td>Manager process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Schedule process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Inventory delivery process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Inventory edit process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Auto decrement of inventory</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Reports</td>
<td>Limited</td>
<td>More advance</td>
</tr>
<tr>
<td>Order Taking Process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Dispatch orders process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Cash out orders process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Create drivers</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Cash out driver</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Unassign orders</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 14.1: Table of Pizza Hut Features comparisions.

One of the big companies that developed applications for restaurant and retail stores is PCMS Datafit. PCMS Datafit develops Java based application and the Pizza Store would be in competition for the same clients. PCMS Datafit develops their applications in two parts, the back office where the administration and manager tools are developed, and the POS (Point Of Sale) where the orders are taken. The back office is developed as a Java J2EE application and the POS is developed as Java AWT and Swing application.

As you can see below, the PCMS Datafit application and the Pizza Store application has similar features and some differences between the features. The PCMS Datafit system was developed for both a J2EE application and a Java AWT/Swing application. The POS part of the system was developed as a standalone Java applet application. The applet has to be installed on every terminal that needs to use the application. The Pizza Store is a full web based application and can be used by any computer that has a web browser. In the event of computer terminal failures, there are no additional installations required. Since the POS
is a Java applet application the computer has to setup an ODBC connection through the operating system. Since the Pizza Store is a web application and does not need setup an ODBC connection. The only minor difference is the POS is set up to be a touch screen. The Pizza Store, although not developed for touch screen in mind, it can utilize the touch screen to some degree. Figure 14.2 is a table of PCMS DataFit Features and Pizza Store comparisons.

<table>
<thead>
<tr>
<th>Features</th>
<th>PCMS DataFit</th>
<th>Pizza Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display</td>
<td>Browser, Java app</td>
<td>Browser</td>
</tr>
<tr>
<td>Navigation</td>
<td>Mouse or touch</td>
<td>Mouse</td>
</tr>
<tr>
<td>Server Connection</td>
<td>Web server/Loca</td>
<td>Web server</td>
</tr>
<tr>
<td>Administrator process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Clocking in and out process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Logon process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Logon as role-user</td>
<td>as user with a role</td>
<td>as user with a role</td>
</tr>
<tr>
<td>Manager process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Schedule process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Inventory delivery process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Inventory edit process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Auto decrement of inventory</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Reports</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Order Taking Process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Dispatch orders process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Cash out orders process</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Create drivers</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Cash out driver</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Unassign orders</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 14.2: Table of PCMS DataFit Features comparisons.

One option for a free open source shopping cart application is Zen Cart. Zen Cart is a shopping cart application used to make it easier for a user to set up a store and take orders. As you can see below the Zen Cart application and the Pizza Store application have some similar features, but most of the features of the Pizza Store the Zen Cart doesn’t have. The features that the Zen Cart does have, is limited in comparison to the Pizza Store features. The problem is the Zen Cart is just a shopping cart application and is only good for the order taking process. It is limited in the Administration process because it lacks the employee, inventory and user roles administration. It is limited in the Manager task because it is lacks the ability to manage employees and inventory. It is limited in the reporting feature because it can only produce reports for products sold and customer database. It doesn’t even have the capacity for dispatching orders and maintaining drivers. Although the Zen Cart is strong in the order taking process, it lacks the essentials for running a pizza delivery store. Figure 14.3 is a table of Zen Cart Features and Pizza Store comparisons.
14.4 Market Analysis Other Applications

Below is a list other applications available in the market place. The list provides a brief description on the application flaws in comparison to the Pizza Store application.

1. Cash Register: http://keyhut.com/pos.htm - It is a DOS base application and must be installed on every computer that needs to use it.

2. CAM Commerce Solutions - Retail ICE - has similar functionality, but it is used for retail sales and not restaurant sales.

3. Positive Feedback Software (for restaurants) is a develop in PHP
   a. It used for retail sales and not restaurant sales.
   b. Java Application servers enforce security model. PHP does not.
   c. Java/JVM has its own extensive security framework. PHP does not.
   d. No security guarantees from PHP.

4. PHP Point of Sale: Is a PHP based application same as 3.

5. L’ane POS: Is not a web application. It seems to be a Perl application that would need to be installed on every computer that needs to use it.

6. MozPOS: it is used for retail sales and not restaurant sales.

Table 14.3: Table of Zen Cart Features comparisions.
7. FreePOS: Is a PHP based application same as 3.

8. IS4C (for Co-ops): Is a PHP based application same as 3.

9. Action Point of Sale: Seems to be just a point of sale or cash register application.

10. TurboCASH: Is an Accounting Software.

11. Free Delivery and Takeout Pizza POS Software: - seems to be just a POS application and is not developed as a web application.

12. Posterita free web based POS: - seems to be just a POS application and is not developed as a web application. A third party websites can connect to the POS application, but it is not a true web application.


14. ProffittCenter: - Seems to be just a POS application and is not developed as a web application.

15. Rapidpos Pizza-Shop-Point-Of-Sale was develop on a Microsoft platform and, as I discuss in my paper, would cost the customer more money to deploy the application on Microsoft Servers.

16. http://www.pizzasoftware.com/ - is a POS and is not developed as a web application.
Chapter 15

Conclusion

As discussed throughout the paper, the design and development of the application was to provide a functional web application for the needs of a small to medium restaurant business. The application was developed to provide for those needs of the business with a minimum cost to the business. The paper explored the Open Source server environment of Ubuntu, Apache Tomcat, and Mysql servers. The paper showed that Ubuntu, Apache Tomcat, and Mysql servers had the capability to support the current and future needs of a small to medium size restaurant business, at a fraction of the cost of other server systems. The paper examined the J2EE platform and Struts framework, and explained the benefits that they provided in developing the application. The paper reviewed the design and development of the application’s 6 functionality modules: ClockIn/Clockout, Logon, Administrator, Manager, Order, and Casher/Dispatch. The paper review the JSP pages that were designed and developed to be utilized by the users of the application. The paper discussed the design and development of the Java Class used in the back of the application. The paper finally discussed the design and creation of the database schema.
Appendix A

index.jsp

Below is a sample code of index.jsp

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>

<html>
    <head>
        <%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"
        <response.setHeader("Cache-Control", "no-cache");
        response.setHeader("Cache-Control", "no-store");
        response.setHeader("Pragma","no-cache");
        response.setDateHeader("Expires", 0);%>
        <response.setHeader("Cache-Control", "no-cache");
        response.setHeader("Cache-Control", "no-store");
        response.setHeader("Pragma","no-cache");
        response.setDateHeader("Expires", 0);%>
        ServletContext sc = getServletConfig().getServletContext();
        MiscUtilBean mub = new MiscUtilBean();
        String contextUrl = mub.getString(sc.getAttribute("contextUrl"));
        ErrorMsgBean emb = new ErrorMsgBean();
        emb.setErrCase(request.getParameter("errCase"));
        emb.setMsgStyleF("msgFailedStyle");
        emb.setMsgStyleS("msgSuccessStyle");
        emb.setMb((MessagesBean)sc.getAttribute("messagesBean"));
        emb.setDs((MysqlDataSource)session.getAttribute("ds"));
        emb.initMsg();%>
        String urlPage = "logon.jsp";
        String header = "/_includes/header.jsp";
        if(request.getParameter("page") != null)
            urlPage = request.getParameter("page") + "\.jsp";
        urlPage += ".jsp";%>
        <META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
        <META name="GENERATOR" content="IBM Software Development Platform">
        <META http-equiv="Content-Style-Type" content="text/css">
```
<TABLE WIDTH="100%" HEIGHT="100%" CELSPACING="0" CELLPADDING="0" BORDER="0">
  <TR VALIGN="TOP">
    <TD COLSPAN="2" CLASS="HEADERSTYLE">
      <JSP:INCLUDE PAGE="<%= header%>" FLUSH="true"/>
    </TD>
  </TR>
  <TR VALIGN="TOP">
    <TD COLSPAN="2" CLASS="CONTENT">
      <JSP:INCLUDE PAGE="<%= urlPage%>" FLUSH="true"/>
    </TD>
  </TR>
  <TR>
    <TD COLSPAN="2" CLASS="FOOTERSTYLE">
      <JSP:INCLUDE PAGE="/included/footer.jsp" FLUSH="true"/>
    </TD>
  </TR>
</TABLE>

<DIV ID="calendarDiv1" STYLE="POSITION:ABSOLUTE;VISIBILITY:HIDDEN;BACKGROUND-COLOR:WHITE;LAYER-BACKGROUND-COLOR:WHITE;"/>
</DIV>
References


urlhttp://www.ubuntu.com/project/about-ubuntu/licensing.


urlhttp://download.oracle.com/javaee/1.3/api/javax/servlet/Servlet.html.


urlhttp://dev.mysql.com/tech-resources/articles/dispelling-the-myths.html.

urlhttp://www.w3schools.com/DTD/default.asp.
    urlhttp://www.w3schools.com/js/default.asp.

[17] w3schools. Xml schema tutorial.
    urlhttp://www.w3schools.com/schema/default.asp.

[18] w3schools. Xml tutorial.
    urlhttp://www.w3schools.com/xml/default.asp.


    urlhttp://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller#cite_note-Reenskaug-0l.

    urlhttp://en.wikipedia.org/wiki/MySQL.