Foods R Us

An E-Commerce Website

Designed and Developed By
Adam Adjindji (cse: adamziz)
Daniel Yakubov (cse: daniely9)
Ragheb Abunahla (cse: ragheban)

Submitted as the group project for the course
EECS4413 - Building E-Commerce Systems
Prof. H. Roumani - Fall 2018

Table of Contents:

Design:

System Architecture Description

The system architecture is modelled over the project D architecture, where the model, view and control are all separated in their concerns. Additionally, this means that the view is a .JSP based view. Our project contains no dynamic alterations via JavaScript.

A client would log in using their EECS log in ID which makes a request from the Roumani server. It is secured since we are not exposing any return parameters from the server. Additionally, instead of forwarding we only redirect to where the client was (as a means to not give out any information such as the username, hash, etc.).

The client makes a request and that request is handled by the controller. The controller understands what each button and entry would mean from the users and may call an *Engine* class in the model that is responsible for processing any computation, analytics and data access. Under certain scenarios, the controller may also request a user to sign in.

The model processes a variety of computations and among which include:

- Data Access Objects to make SQL queries form the database to fill Java Beans for the items and categories.
- Java Beans that allow the controller and the view to access bundled information effectively for the Customer, Order, Item and Category.
- Any mathematical computations such as calculating costs, sales tax, delivery fees, logical computation (in the *Engine*).
- Processing orders and creating PO XML files (in the *Engine*) as well as file creation and disk folder access.
- Error handling and checking and will throw exceptions with error messages if an error was to come up.

Once the model is done with its duties, it sends the required information back to the controller which sets attributes according depending on user requests. The view takes in these alterations and upon reloading the pages via the controller, populates the webpage

responsibly with the alterations. The controller lets the view know when to display errors by letting the page know.

There also exists a middleware within a seperate project that takes all PO XML files in an inbox folder and extracts the total quantities from all the XML files which is converted into a report file. The report is stored in the root of the PO folder. All the processed order XML files are then put in an outbox folder to note that it is done. The middleware takes into account concurrency so that files are not in the inbox and outbox at the same time by …

In terms of namespaces, we we adamant on giving a certain format name for all the variable IDs, classes, buttons and method such that the process was intuitive to developers and editors. For example, the `input` of type submit on JSP was named "addToCartButton" and the `div` containing it was named "addToCartDiv" and the `form` containing it was called "addToCartForm", etc. Additionally, our class and filenames were consisting throughout the project. So if a servlet was called 'Cart.java', then the JSP file associated with it was called 'Cart.jspx'.

Design Issues and Decisions

The JSP approach to the project was taken as opposed to using plain JS or a framework for the View due to its cleaner separation of concerns and the experience level of our team.

The following are our design problems and how we resolved them:

Problem: In what data structure or type to store the Cart in for easy alterations and viewing.

Decision: We decided to store the cart in two ways depending on the purpose. We store it in a Map<ItemBean, Integer> whenever the user wants to see the cart or see the items so that the view can access ItemBean methods such as *getName(), getPrice(), etc.* and the Integer is the quantity the user wants to buy. In the session, the cart is stored as a Map<String, Integer> where the String is the item ID/number and the Integer is the quantity the user wants to buy. The distinction here is that the one only with the item number is stored in the session to (1) to reduce the amount of data we store in the session (2) if a user want to delete an item

from the cart, it can just search for the item ID in the key as opposed to the item bean in memory.

- - -

Problem: Where to store the Cart items such that we can easily pull it whenever the user wants to append it view it.

Decision: We decided to store the cart in the sessions scope so that it can be retrievable by just requesting the attribute name from the HttpSession in any controller we needed it.

- - -

Problem: Designing concurrency within the middle ware.

Decision: Middleware is implemented as a single java class. The class expects the path of the PO directory as it's program argument. Numerous error checking are performed to ensure that the argument is a directory with the required inPO and outPO folders created by the web app. The middleware also exits quickly if the inPO folder is empty, saving time and resources.

- - -

Problem: How the filter was going to provide advertisements without altering any WebApp features like the Model, View and Control.

Decision: We decided to create a seperate ad-hoc folder from our architecture and also create another JSP page to accompany the Java filter. This additionally JSP page mimics the cart exactly, but add a `div` that shows the cross referenced and recommended item.

- - -

Problem: Storing and deciding on which beans we needed for our design.

Decision: We created an ItemBean because of the database. We created a CategoryBean for our Browse feature for dynamic viewing and so we can extract the image from it. We created a CustomerBean for to store the information of a customer within the session and assign them as cart while they’re signed in. We created a OrderBean to process the orders and marshal/un-marshal the XML file.

- - -

Problem: Making the PO files only accessible to users who made the orders.

Decision: Users may not see POs until they are signed in, additionally there is regular expression functionality that makes sure the corresponding user sees the correct PO.

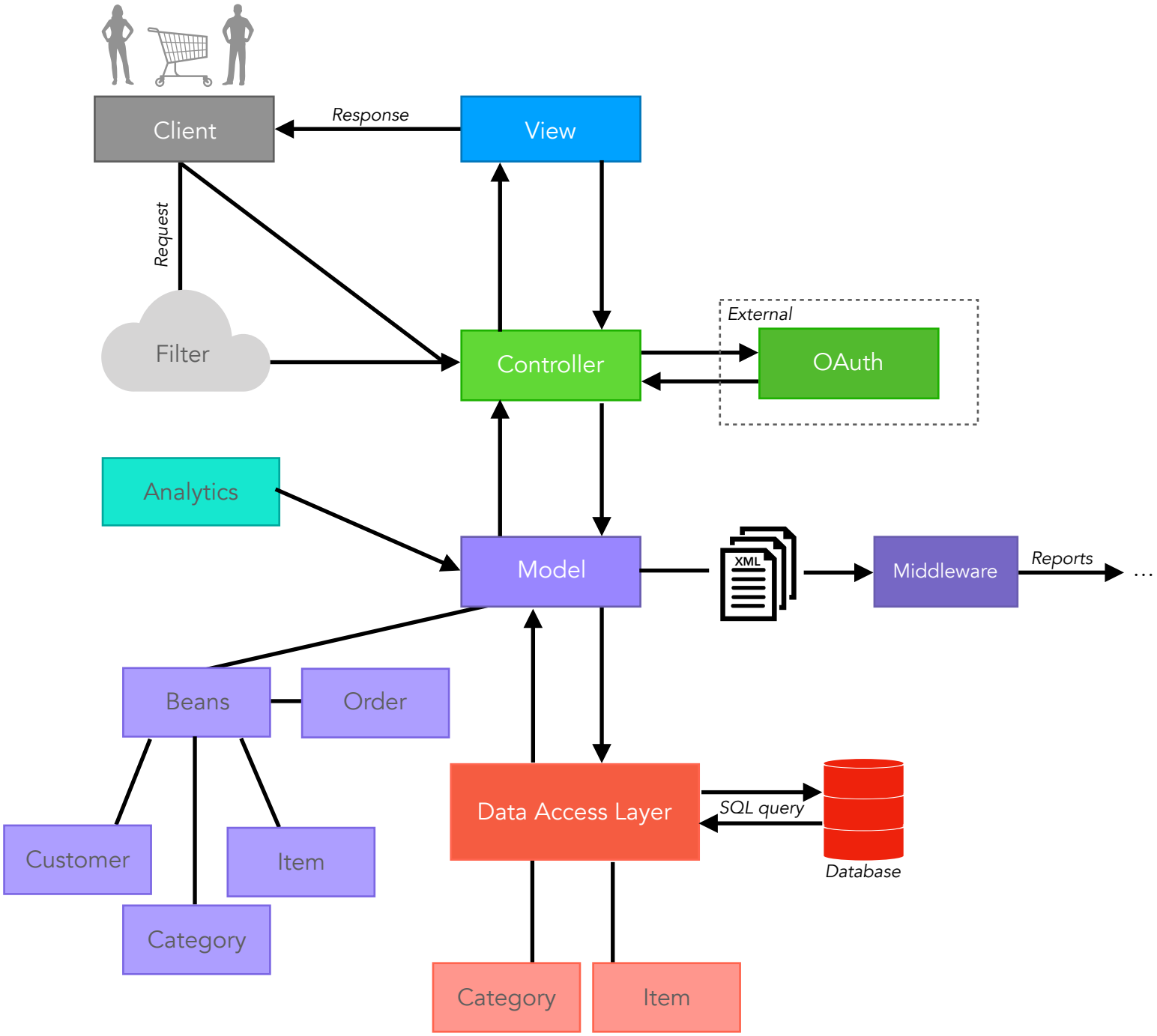
Testing and Status

In terms of testing, we have used JUnit testing for the Model and we ran the website onto the server several times and printed out errors if need be. We purposely try to break the website on multiple occasions to see where improvements and fixes were needed. We tried to be regular users.

Below is the completed status of the project accomplishments based the requirements:

Catalog servlet display items and the corresponding categories
Client can add items to empty cart
Cart servlet shows the cart items in the tabular format
Cart servlet displays number, name, extended price and unit price of each item in a read-only fashion
Cart servlet should have a writable quantity section with functionality
Cart servlet should include shipping, HST, total cost and summed costs
Cart servlet has three buttons: Update, Continue Shopping and Checkout
A client is logged in by redirecting to AUTH whether prompted to or wants to
A client can checkout and is given a display screen of approval
A client is also given a set of link or links based on their most recent order or all their orders to view
P/O files are in the XML format and follow the appropriate schema and naming convention for the user to view.
Management average time can be viewed via a mechanism
Ad-hoc recommended items are provided in the cart via a filter
A client can search items at any point on the website.
Improved security is ensured
Middleware concurrency is ensured (asynchronous)

Data Flow Diagram:



Implementation Issues and Decisions

The following are out design problems and how we resolved them:

Problem: Implementation of recommendation analytics and how recommendations were made.

Decision: A map of items associated with their recommended items that perhaps a business team would have decided on. Based on this map, if the user goes to their Cart and the filter catches any items that exist in the recommendation map, they would be sent to an alternative cart with the recommendation and an option to add it. This is easily removable by just turning the filter off.

- - -

Problem: Algorithm for middleware.

Decision: The middleware works by generating the required collections and final report through methods build on each other. First the po.xml files are put into a list by the first method. Then this list is used as an argument in the second method to generate a map with the item number as key, and a new bean that contains the item number, name, and total quantity from all the orders. This map is then used in the third method to generate the reportBean. Finally, the report bean is put into the final method which marshalls the report into a report.xml file numbered by the amount of reports and placed in the PO root folder, and calls a private method to move the processed orders into the outPO folder.

The middleware was designed to be extensible, but also light and in a single class with helper java beans. The ReportBean class can easily be extended with additional data, and changing the output is decoupled from the rest of the logic. The middleware requires a jar file containing the necessary beans from the web app to process the xml files into it.

- - -

Problem: How to keep track of which item to add to the cart in a page full of several items and several “Add to Cart” buttons?

Decision: We added an input of type hidden that is associated to each item so we can extract it in the controller and pass that information to the model to add to the session cart.

- - -

Problem: Since the cart is stored in the session, how does the model engine add to it?

Decision: We send the reference of the cart from the controller as a parameter of the engine method and the engine returns an updated cart that the controller gets and updates in the session.

- - -

Problem: How do we make sure that the sort by and category selection option in the Browse page remain the same when a user adds an item to the cart?

Decision: We store the 'sort by' and 'category' information as an attributes and pass it back with the request so that the user is still viewing the same preferences they started with.

- - -

Problem: How do we implement analytics

Decision: We created a list in the server context per action and the item in each list is the time it took the user to perform the respective action. For instance for the "Add to Cart" list, whenever a user starts a session we record the start time, so that when they complete add an item to the cart, we can find the total time that action took. We also did this for a "Checkout" list. When the admin wants to view these statistics, they go to the Admin.do page and view these statistics in a tabular format in seconds.

The Team

Team Establishment

This portion of the project was taken very seriously so that the project could have a smooth sailing workflow, the members could be held accountable and so that the work could be distributed equally. We also set of clear expectations of how we wanted to work and what our personal deadline was for the fundamental functionality.

Organizational Details

The team had an initial meeting to go over the group project requirements line-by-line and discuss implementation. Those meeting minutes can be found here: https://docs.google.com/document/d/1jO3vVk6uDp0gqmSkmezl6ChC2mNuUZ3uEsA5lOubwhc

We decided to run a multiple-hat-wearing work approach, such that no single person was responsible for model or view, etc. Each person on the team got exposure, to a variety of different responsibilities and project requirements, whether it be the same file or different files.

From then onward, the team met on almost daily basis from November 16th and onwards in order to compete the project. At every work session, the team gathered to set out targets for the day and would check in at the end to (1) assess what has been completed so far (2) discus when they were to meet next. About 80-90% of the time work happened when all team members were present.

The team also set up a GitHub (LINK) and has committed over 200 times with comments explaining each commit, so that they could retrace and refactor accordingly. There were multiple branches based off of the middleware, view, control, etc. Commits and pushes were only made to the Master when the whole team reached a consensus.

Lessons Learned & Thoughts in Hindsight

Theres a couple things we would do differently if we had to start from scratch. We would firstly start this report as we make progress as opposed to starting it before the last day. We also had trouble adding some implementation when the styling was lacking, so we ended up styling the website fairly late as opposed to doing it as we build.

Nonetheless, we also found that JUnit testing for Engine functionality was fairly useful with a lack of view being built. We thought the agile/scrum methodology we followed while working allowed us to really benefit in terms of progress and have all the team members remain on the same page. In retrospect, although the GitHub was useful, it would have been better to establish expectations so that fixed changes would have already been committed. We enhanced our understanding of how to separate concerns.

Individual Paragraphs and Work List

Ragheb Abunahla

Role Played: Project organizer, controller and functionality development and website design. I received an immense amount of experience understanding how a project management experience is like, additionally I got a lot of exposure to working out user experience in terms of website functionality and expectations. I also learned a lot about the logic and implement of the Controller. since I found myself mostly focused there, often communication and meddling with the View code. I had some experience with the Model's Engine as I've had to make some methods to test my Controller responses.

I hereby attest to the accuracy of the information contained in "The Team" section of this Report.

Adam Adjindji

Role Played: Designed middleware, DAL, Beans, analytics, XML files and folders, git organizer. I spent time designing the middleware in terms of extensibility and speed. The beans were implemented to be as simple as possible to use for both calculations and manipulation in the model, as well as handling data to and from XML files. Analytics were implemented to take advantage of context storage, and a simple Admin.do page was provided to view the information. I encouraged the use of git, github, and feature workflow. This was invaluable when changes were introduced that broke key functionality. I learned about the work of my team members by adding new functionality and fixes to the sections that they worked on, as well as our explanations of our implementations in our daily meets. Initially I worked almost entirely on the model and DAO, but in the late stages of the project I spent far more time in the controllers and view. I understand every aspect of the project and it's implementation.

I hereby attest to the accuracy of the information contained in "The Team" section of this Report.

Daniel Yakubov

Role Played: Front end and Filter Developer. My major contribution was to develop the lay out of the website, from the navbar to the styling of each component From this I leaned bootstrap CSS rather well. Also I developed the filter, this process required the use of most of the control classes. This allowed for a functional well rounded ad-hoc. Although I learned a lot my self, what I learned from my team mates was above my expectation. Rae was a well composed project manager that made sure we kept on top of the project, from creating to-do lists and suggesting the communication channels. Rae helped me a lot though out the process of the layout, he created diagrams of the site for me to implement. Rea contributed on all aspects of the project and made sure our team was up to speed with everything he implemented on the cart and most of the controllers. Adam's insights on the project helped us create a well secured thought out back-end. He also suggested that we use GitHub, this has made me very confident with the platform. Finally I have to say I feel confident that I could have learned more about listeners and SessionContext. Although for the majority of the project I feel very comfortable and feel confident to make changes in a test environment

I hereby attest to the accuracy of the information contained in "The Team" section of this Report.

Table of work distribution

	Adam Adjindji	**Ragheb Abunahla**	**Daniel Yakubov**
View (.jspx)			
Account			
Cart			
Catalog			
Checkout			
Dash			
Order			
Search			
Admin			

	Adam Adjindji	Ragheb Abunahla	Daniel Yakubov
ViewOrder			
Control (.java)			
Account			
Auth			
Cart			
Catalog			
Checkout			
Dash			
Order			
Search			
ViewOrder			
Model (.java)			
Engine			
CategoryBean			
CategoryDAO			
ItemBean			
ItemDAO			
CustomerBean			
OrderBean			
Listeners (.java)			
Authentication			
CartInit			
Init			
Middleware (.java)			
Middleware			
Filter (.java)			
Reference			

Source Code

View

Account.jspx

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:c="http://java.sun.com/jsp/jstl/core"
xmlns:fmt="http://java.sun.com/jsp/jstl/fmt" version="2.0">
<jsp:directive.page contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" session="true" />
<jsp:output doctype-root-element="html"
doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
omit-xml-declaration="true" />

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Account - Foods R Us</title>
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/
bootstrap.min.css"
integrity="sha384-MCw98/
SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous" />
<link href="css/style.css" rel="stylesheet" type="text/css" />
</head>
<body class="container" background="img/backdrop2.png">
<div><br/></div>
<!-- STRAT OF NAVIGATION -->
<nav class="navbar navbar-expand-lg navbar-light rounded"
style="background-color: #f8b39d;">
<!-- START OF LOGO -->
<div class="navbar-brand col-md-2">
```

```
<a href="Dash.do" name="home"> <img src="img/efoods.png"
class="img-circle" width="60" height="60" alt="" />
</a>
</div>
<!-- SEARCH -->
<form name="searchForm" action="Search.do" method="POST"
class="form-inline my-2 my-lg-0">
<input class="form-control " type="search" style="width: 375px"
name="searchInput" placeholder="Search" value="${searchInputValue}">
<button class="btn btn-outline-primary my-2 my-sm-0" type="submit"
name="searchButton" value="Search">Search</button>
</input>
</form>
<!-- LINKS -->
<ul class="navbar-nav ml-auto">
<!-- CATALOG -->
<li class="nav-item active"><a class="nav-link"
href="Catalog.do" name="browse"> Browse </a></li>
<!-- CART -->
<li class="nav-item nav-a nav-a-2 active"><a class="nav-link"
href="Cart.do"><small>Your,</small><br />Cart <small
class="badge badge-pill badge-danger">${cart.size()}</small></a></li>
<!-- SIGN IN -->
<li class="nav-item active">
<form>
<c:if test="${!sessionScope.authenticated}">
<a class="nav-link" href="Auth.do" name="authentication">Sign
In</a>
</c:if>
<c:if test="${sessionScope.authenticated}">
<a class="nav-link" href="Account.do" name="account"><small>Hello,
${sessionScope.customer.getName().split(" ")[0]} </small><br />Your
```

```
account</a>
</c:if>
</form>
</li>
</ul>
</nav>
<div name="mainDiv" class="col-md-12 rounded" style="background-color:
#fffdfc; height: 100%;">
<br/>
<h1>My Account:</h1>
<p>Username: ${sessionScope.customer.getAccount()}</p>
<p>Full Name: ${sessionScope.customer.getName()}</p>
<div name="orderListDiv">
<h4>Past Orders:</h4>
<c:forEach var="order" items="${sessionScope.previousOrders}">
<p>
<a href="ViewOrder.do?orderName=${order.key}" target="_blank">$
{order.key}</a>
</p>
</c:forEach>

<br/>
</div>
</div>
</body>
</html>
</jsp:root>
```

Admin.jspx

```
<?xml version="1.0" encoding="UTF-8" ?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:c="http://java.sun.com/jsp/jstl/core"
```

```
xmlns:fmt="http://java.sun.com/jsp/jstl/fmt" version="2.0">
<jsp:directive.page contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" session="false" />
<jsp:output doctype-root-element="html"
doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
omit-xml-declaration="true" />
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Admin</title>
</head>
<body>
<c:set var="addSessions" value="${applicationScope.timeBetweenAdd}"></
c:set>
<c:set var="checkoutSessions"
value="${applicationScope.timeBetweenCheckout}"></c:set>

<div class="analytics">
<h3>Average time user took to add an item</h3>
<c:if test="${addSessions.size() gt 0}">
<p>${addSessions.size()} user(s) took ${averageAdd} seconds</p>
</c:if>
<c:if test="${addSessions.size() eq 0}">
<p>${averageAdd}</p>
</c:if>
</div>
<hr />
<div class="analytics">
<h3>Average time user took to checkout</h3>
<c:if test="${checkoutSessions.size() gt 0}">
<p>${checkoutSessions.size()} user(s) took ${averageCheckout} seconds</p>
</c:if>
```

```
<c:if test="${checkoutSessions.size() eq 0}">
<p>${averageCheckout}</p>
</c:if>
</div>

</body>
</html>
</jsp:root>
```

Cart.jspx

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:c="http://java.sun.com/jsp/jstl/core"
xmlns:fmt="http://java.sun.com/jsp/jstl/fmt" version="2.0">
<jsp:directive.page contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" session="true" />
<jsp:output doctype-root-element="html"
doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
omit-xml-declaration="true" />

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title> Cart - Foods R Us </title>
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/
bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/
SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous" />
<link href="css/style.css" rel="stylesheet" type="text/css" />
</head>
<body class="container" background="img/backdrop2.png">
<!-- STRAT OF NAVIGATION -->
```

```
<div>
<br/>
</div>
<nav class="navbar navbar-expand-lg navbar-light rounded-top"
style="background-color: #f8b39d;">
<!-- START OF LOGO -->
<div class="navbar-brand col-md-2">
<a href="Dash.do" name="home" >
<img src="img/efoods.png"  class="img-circle" width="60" height="60"
alt="" />
</a>
</div>
<!-- SEARCH -->
<form name="searchForm" action="Search.do" method="POST" class="form-
inline my-2 my-lg-0">
<input class="form-control " type="search" style="width:375px"
name="searchInput" placeholder="Search" value="${searchInputValue}">
<button class="btn btn-outline-primary my-2 my-sm-0" type="submit"
name="searchButton" value="Search">Search</button></input>
</form>
<!-- LINKS -->
<ul class="navbar-nav ml-auto">
<!-- CATALOG -->
<li class="nav-item active">
<a class ="nav-link" href="Catalog.do" name="browse"> Browse </a>
</li>
<!-- CART -->
<li class="nav-item nav-a nav-a-2 active">
<a class ="nav-link" href="Cart.do"><small>Your,</small><br />Cart <small
class="badge badge-pill badge-danger">${cart.size()}</small></a>
</li>
<!-- SIGN IN -->
<li class="nav-item active">
<form>
<c:if test="${!sessionScope.authenticated}">
```

```
<a class ="nav-link" href="Auth.do" name="authentication">Sign In</a>
</c:if>
<c:if test="${sessionScope.authenticated}">
<a class ="nav-link" href="Account.do" name="account"><small>Hello, $
{sessionScope.customer.getName().split(" ")[0]} </small><br />Your
account</a>
</c:if>
</form>
</li>
</ul>
</nav>
<div class="cartDiv rounded col-md-12" style="background-color:
#fffdfc;">
<br/>
<h2>Your Cart</h2>

<div class="cartItemTableDiv" >
<c:if test="${viewableCart.isEmpty()}">
<p> Your cart is empty. Add items <a href="Catalog.do"> here. </a> </p>
</c:if>
<c:if test="${!viewableCart.isEmpty()}">
<form name="updateCartForm" action="Cart.do" method="POST">
<table class="table">
<tr class="font-weight-bold">
<td> Name </td>
<td> Quantity </td>
<td> Cost Per Unit </td>
<td> Price </td>
<td> ID </td>
<td> Delete </td>
</tr>
<c:forEach items="${viewableCart}" var="cartItem">
<input type="hidden" name="itemId" value="${cartItem.key.getNumber()}"/>
```

```
<tr>
<td>${cartItem.key.getName()}</td>
<td>x <input type="number" min="0" step="1" value="${cartItem.value}"
name="quantityInput"/></td>
<td>CAD<fmt:formatNumber  type="currency" value="$
{cartItem.key.getPrice()}" /></td>
<td>CAD<fmt:formatNumber  type="currency" value="$
{cartItem.key.getPrice() * cartItem.value}" /></td>
<td>${cartItem.key.getNumber()}</td>
<td> Delete: <input type="checkbox" value="${cartItem.key.getNumber()}"
name="deleteCheckbox"/> </td>
</tr>
</c:forEach>
</table>
<div class="text-right">
<button class="btn btn-outline" type="submit"
name="updateCartButton">Update</button>
</div>
<c:if test="${!empty error}">
<p style="color: red;"> <strong> ${error}</strong></p>
</c:if>
</form>
</c:if>
</div>
<div class="checkoutDiv">
<form name="checkoutForm" action="Checkout.do" method="POST">
<h2> Price Calculations: </h2>
<p name="itemsCost"> Cost of Items: CAD<fmt:formatNumber type="currency"
value="${itemsCost}" /> </p>
<p name="hstAmount"> HST: CAD<fmt:formatNumber type="currency" value="$
{hstAmount}" /> </p>
<p name="shippingCost"> Shipping Cost: CAD<fmt:formatNumber
type="currency" value="${shippingCost}" /> </p>
<p name="overallCost"> Total Cost: CAD<fmt:formatNumber type="currency"
value="${shippingCost+(itemsCost+hstAmount)}" /> </p>
```

```
</form>
<div class="row">
<form name="checkoutForm" action="Checkout.do" method="POST">
<input class="btn btn-outline m-1 col" type="submit"
name="checkoutButton" value="Checkout"/>
</form>
<div class="col-2" />
<form name="checkoutForm" action="Catalog.do" method="POST">
<input class="btn btn-outline m-1 col" type="submit"
name="continueShoppingButton" value="Continue Shopping"/>
</form>
</div>
</div>
<br/>
</div>
</body>
</html>
</jsp:root>
```

Catalog.jspx

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:c="http://java.sun.com/jsp/jstl/core"
xmlns:fmt="http://java.sun.com/jsp/jstl/fmt" version="2.0">
<jsp:directive.page contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" session="true" />
<jsp:output doctype-root-element="html"
doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
omit-xml-declaration="true" />

<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>


<title> Catalog - Foods R Us </title>
<meta name="viewport" content="width=device-width, initial-scale=1" />
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/
bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/
SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous" />                    <link href="css/style.css"
rel="stylesheet" type="text/css"></link>
<link href="css/style.css" rel="stylesheet" type="text/css" />


</head>
<body class="container" background="img/backdrop2.png">
<div>
<br/>
</div>
<!-- STRAT OF NAVIGATION -->
<nav class="navbar navbar-expand-lg navbar-light rounded-top container"
style="background-color: #f8b39d;">
<!-- START OF LOGO -->
<div class="navbar-brand col-md-2">
<a href="Dash.do" name="home" >
<img src="img/efoods.png"  class="img-circle" width="60" height="60"
alt="" />
</a>
</div>
<!-- SEARCH -->
<form name="searchForm" action="Search.do" method="POST" class="form-
inline my-2 my-lg-0">
<input class="form-control " type="search" style="width:375px"
name="searchInput" placeholder="Search" value="${searchInputValue}">
<button class="btn btn-outline my-2 my-sm-0" type="submit"
name="searchButton" value="Search">Search</button></input>
</form>
<!-- LINKS -->
```

```
<ul class="navbar-nav ml-auto">
<!-- CATALOG -->
<li class="nav-item active">
<a class ="nav-link" href="Catalog.do" style="float-bottom;"
name="browse"> Browse </a>
</li>
<!-- CART -->
<li class="nav-item nav-a nav-a-2 active">
<a class ="nav-link" href="Cart.do"><small>Your,</small><br />Cart <small
class="badge badge-pill badge-danger">${cart.size()}</small></a>
</li>
<!-- SIGN IN -->
<li class="nav-item active">
<form>
<c:if test="${!sessionScope.authenticated}">
<a class ="nav-link" href="Auth.do" name="authentication">Sign In</a>
</c:if>
<c:if test="${sessionScope.authenticated}">
<a class ="nav-link" href="Account.do" name="account"><small>Hello, $
{sessionScope.customer.getName().split(" ")[0]} </small><br />Your
account</a>
</c:if>
</form>
</li>
</ul>
</nav>
<!-- MAIN DIV -->
<div name="mainDiv" style="background: #fffdfc;">


<br />
<!-- TITLE -->
<div name="catalogDiv" class="m-2">
```

```
<h2>Select a Catalog</h2>
</div>
<br />
<!-- CAT ITEM BUTTONS -->
<div class="row justify-content-md-center">
<c:forEach items="${catalogList}" var="catalogItem">
<div class="col-sm-3 text-center " style="color: #fffdfc;">
<div class="card m-1" style="background-color: #fffdfc;">
<div class="card-header" style="background-color: #f8b39d;">
${catalogItem.getName()}
</div>
<form name="catalogForm" action="Catalog.do" method="POST">
<input type="image" name="submit" src="$
{catalogItem.getPicture()}" />
<input type="hidden" value="${catalogItem.getId()}" name="catalogId"/>
<div class="card-body" style="background-color: #f8b39d; ">
<small class="" style="color: white;">
${catalogItem.getDescription()}
</small>
</div>
</form>
</div>
</div>
</c:forEach>
</div>
<br />
<!-- SORT OPTIONS -->
<div name="sortByDiv" class="text-right m-3 ">
<form name="catalogOrder" action="Catalog.do" method="POST">
<select id="sortBy" name="sortBy" class="m-2  ">
<option selected="any" value="NONE">Select One:</option>
<option value="A to Z">A to Z</option>
```

```
<option value="Z to A">Z to A</option>
<option value="Price - Low to High">Price - Low to High</option>
<option value="Price - High to Low"> Price - High to Low</option>
</select>

<button type="submit" name="sortByButton" value="Sort" class="btn btn-
outline">Sort</button>
<input type="hidden" value="${catalogId}" name="catalogId"/>
</form>
</div>

<!-- SortedBy -->
<div name="showingCategoryDiv" class="border-bottom text-right m-3">
<h6>
Showing: ${selectedCatalogName}
<br />
Sorted by: ${sortBy}
</h6>
</div>
<div name="catalogResultDiv" class="">
<div class="row justify-content-center">
<c:forEach items="${itemList}" var="item">
<div class="card m-4 " style="width: 300px;">
<form name="cartItemForm" action="Catalog.do" method="POST">
<div class="m-2">
Name: ${item.getName()}
<br />
Price: CAD<fmt:formatNumber  type="currency" value="${item.getPrice()}" /
>
<br />
Quantity:
<input type="number" style="float: right" value="1" placeholder="1"
name="addQuantity" step="1" min="1"/>
```

```
<button type="submit" style="float: right" name="cartButton" class="btn
btn-outline m-2">Add to Cart</button>
<br />
ID: ${item.getNumber()}
<c:if test="${cart.containsKey(item.getNumber())}">

<p style="color: green;"> ${cart.get(item.getNumber())} of this item
exists in the cart. </p>
</c:if>
<input type="hidden" name="hiddenItemBeanId" value="${item.getNumber()}"/
>
<input value="${sortBy}" name="sortBy" type="hidden"/>
<input value="${catalogId}" name="catalogId" type="hidden"/>
</div>
</form>
</div>
</c:forEach>
</div>
</div>
</div>

</body>
</html>
</jsp:root>
```

Checkout.jspx

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:c="http://java.sun.com/jsp/jstl/core"
xmlns:fmt="http://java.sun.com/jsp/jstl/fmt" version="2.0">
<jsp:directive.page contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" session="true" />
<jsp:output doctype-root-element="html"
```

```
doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
omit-xml-declaration="true" />

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title> Checkout - Foods R Us </title>
<meta name="viewport" content="width=device-width, initial-scale=1" />
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/
bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/
SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous" />                    <link href="css/style.css"
rel="stylesheet" type="text/css"></link>
<link href="css/style.css" rel="stylesheet" type="text/css" />

</head>
<body class="container" background="img/backdrop2.png">
<div>
<br/>
</div>
<!-- STRAT OF NAVIGATION -->
<nav class="navbar navbar-expand-lg navbar-light rounded-top"
style="background-color: #f8b39d;">
<!-- START OF LOGO -->
<div class="navbar-brand col-md-2">
<a href="Dash.do" name="home" >
<img src="img/efoods.png"  class="img-circle" width="60" height="60"
alt="" />
</a>
</div>
<!-- SEARCH -->
<form name="searchForm" action="Search.do" method="POST" class="form-
inline my-2 my-lg-0">
```

```
<input class="form-control " type="search" style="width:375px"
name="searchInput" placeholder="Search" value="${searchInputValue}">
<button class="btn btn-outline-primary my-2 my-sm-0" type="submit"
name="searchButton" value="Search">Search</button></input>
</form>
<!-- LINKS -->
<ul class="navbar-nav ml-auto">
<!-- CATALOG -->
<li class="nav-item active">
<a class ="nav-link" href="Catalog.do" name="browse"> Browse </a>
</li>
<!-- CART -->
<li class="nav-item nav-a nav-a-2 active">
<a class ="nav-link" href="Cart.do"><small>Your,</small><br />Cart <small
class="badge badge-pill badge-danger">${cart.size()}</small></a>
</li>
<!-- SIGN IN -->
<li class="nav-item active">
<form>
<c:if test="${!sessionScope.authenticated}">
<a class ="nav-link" href="Auth.do" name="authentication">Sign In</a>
</c:if>
<c:if test="${sessionScope.authenticated}">
<a class ="nav-link" href="Account.do" name="account"><small>Hello, $
{sessionScope.customer.getName().split(" ")[0]} </small><br />Your
account</a>
</c:if>
</form>
</li>
</ul>
</nav>
<div name="mainDiv" class="col-md-12" style="background-color: #fffdfc;">
<br/>
<div id="titleConfirm">
```

```
<h1> Confirm Your Order</h1>
</div>
<div id="finalCartDiv">
<c:if test="${viewableCart.isEmpty()}">
<p> Your cart is empty. Add items <a href="Catalog.do"> here. </a> </p>
</c:if>
<c:if test="${!viewableCart.isEmpty()}">
<table class="table">
<tr class="font-weight-bold">
<td> Name </td>
<td> Quantity </td>
<td> Cost Per Unit </td>
<td> Price </td>
<td> ID </td>
</tr>
<c:forEach items="${viewableCart}" var="cartItem">
<input type="hidden" name="itemId" value="${cartItem.key.getNumber()}"/>
<tr>
<tr>
<td>${cartItem.key.getName()}</td>
<td>x ${cartItem.value}</td>
<td>CAD<fmt:formatNumber  type="currency" value="$
{cartItem.key.getPrice()}" /></td>
<td>CAD<fmt:formatNumber  type="currency" value="$
{cartItem.key.getPrice() * cartItem.value}" /></td>
<td>${cartItem.key.getNumber()}</td>
</tr>
</tr>
</c:forEach>
</table>

</c:if>
</div>
```

```
<div id="checkoutOptionsDiv" class="row">
<c:if test="${!viewableCart.isEmpty()}">
<div class="m-2">
<form name="confirmOrderForm" action="Order.do" method="POST">
<input class="btn btn-outline " type="submit" value="Confirm Order"
name="confirmOrderButton"/>
</form>
</div>
<div class="col m-2">
<form name="editCartForm" action="Cart.do" method="POST">
<input class="btn btn-outline " type="submit" name="editCartButton"
value="Modify Cart"/>
</form>
</div>
</c:if>
</div>
</div>
</body>
</html>
</jsp:root>
```

Dash.jspx

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:c="http://java.sun.com/jsp/jstl/core"
xmlns:fmt="http://java.sun.com/jsp/jstl/fmt" version="2.0">
<jsp:directive.page contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" session="true" />
<jsp:output doctype-root-element="html"
doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
omit-xml-declaration="true" />
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title> Foods R Us </title>
<link rel="icon" src="img/favicon.ico"/>
<meta name="viewport" content="width=device-width, initial-scale=1" />
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/
bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/
SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous" />
<link href="css/style.css" rel="stylesheet" type="text/css" />
</head>
<body class="container" background="img/backdrop2.png">
<!-- STRAT OF NAVIGATION -->
<div>
<br/>
</div>
<div style="color: #FF525;">
<nav class="navbar navbar-expand-lg navbar-light rounded-top"
style="background-color: #f8b39d;">
<!-- START OF LOGO -->
<div class="navbar-brand col-md-2">
<a href="Dash.do" name="home" >
<img src="img/efoods.png"  class="img-circle" width="60pt" height="60pt"
alt="" />
</a>
</div>
<!-- SEARCH -->
<form name="searchForm" action="Search.do" method="POST" class="form-
inline my-2 my-lg-0">
<input class="form-control " type="search" style="width:375px"
name="searchInput" placeholder="Search" value="${searchInputValue}">
<button class="btn btn-outline my-2 my-sm-0" type="submit"
name="searchButton" value="Search">Search</button></input>
</form>
```

```
<!-- LINKS -->
<ul class="navbar-nav ml-auto">
<!-- CATALOG -->
<li class="nav-item active">
<a class ="nav-link" href="Catalog.do" name="browse"> <large>Browse   </
large> </a>
</li>
<!-- CART -->
<li class="nav-item nav-a nav-a-2 active">
<form>
<a class ="nav-link" href="Cart.do"><small>Your</small><br />Cart <small
class="badge badge-pill badge-danger">${cart.size()}</small></a>
</form>
</li>
<!-- SIGN IN -->
<li class="nav-item active">
<form>
<c:if test="${!sessionScope.authenticated}">
<a class ="nav-link" href="Auth.do" name="authentication">Sign In</a>
</c:if>
<c:if test="${sessionScope.authenticated}">
<a class ="nav-link" href="Account.do" name="account"><small>Hello, $
{sessionScope.customer.getName().split(" ")[0]} </small><br />Your
account</a>
</c:if>
</form>
</li>
</ul>
</nav>
</div>

<!-- MAIN DIV START -->
<div name="mainDiv" style="background: #fffdfc;">
```

```
<br />
<br />
<!-- BANNER -->
<div name="featuredDiv" class="col-md-12">
<img src="img/welcomeBanner.png" id="banner" class="rounded"/>
</div>
<br />
<br />
<!-- CONTENT DIV -->
<div name="contentDiv" class="row justify-content-center">
<!-- CATALOG CARD -->
<div name="catalogDiv" class="col-5  rounded" style="background-color:
#f69a7d;">
<div class="text-center" style="color: white;">
<br />
<strong>Select a Category</strong>
</div>
<br />
<div class="text-center border-top">

<ul class="list-group list-group-flush" style="background-color:
#f69a7d;">
<c:forEach items="${catalogList}" var="catalogItem">
<form name="catalogForm" action="Catalog.do" method="POST">
<li class="list-group-item" style="background-color: #f69a7d;">
<button class="btn btn-block btn-outline-transparent" type="submit"
value="${catalogItem.getName()}" name="catalogName">$
{catalogItem.getName()}</button>
<input type="hidden" value="${catalogItem.getId()}" name="catalogId"/>
</li>
</form>
</c:forEach>
</ul>
```

```
<br/>
</div>
</div>
<!-- INBETWEEN DIV -->
<div style="width: 130px">
<br />
</div>

<!-- LAST ORDER CARD -->
<div name="lastOrderDiv" class="col-5  rounded" style="background-color:
#f69a7d;">
<br />
<div class="text-center" style="color: white;">
<strong>Your Last Order</strong>
</div>
<br />
<div class="border-top" >
<br />
<form>
<c:if test="${!sessionScope.authenticated}">
<p style="color: white;"> Please <a href="Auth.do"> sign in </a> to view
your last order.</p>
</c:if>
<c:if test="${sessionScope.authenticated}">
<!-- This is defaulted to say no orders whether the user has made orders
or not // need to fix -->
<c:if test="${empty sessionScope.lastOrder}">
<p style="color: white;"> You have no orders yet.</p>
</c:if>
<c:if test="${!empty sessionScope.lastOrder}">
<p style="color: white;"> You last ordered: </p>
<ul class="list-group list-group-flush">
<div style="color: white;">
```

```
<c:set var="count" value="0"/>
<c:forEach items="${sessionScope.lastOrder.getItems()}" var="item">
<c:if test="${count lt 5}">
<div>${item.getQuantity()} -
${item.getName()} @ CAD<fmt:formatNumber type="currency" value="$
{item.getPriceFormat()}" /></div>
<c:set var="count" value="${count + 1}"/>
</c:if>
</c:forEach>

</div>
</ul>
<br />
<span style="color: white;">TOTAL : CAD<fmt:formatNumber type="currency"
value="${sessionScope.lastOrder.getTotalFormat()}"/></span>
<div style="position: absolute; right: 0; bottom: 0;
color:white;"><small>Showing at most 5 unique items</small></div>
</c:if>
</c:if>
</form>
</div>
</div>
</div>

<div>
<br/>
<br/>
</div>

</div>
</body>
</html>
</jsp:root>
```

Order.jspx

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:c="http://java.sun.com/jsp/jstl/core"
xmlns:fmt="http://java.sun.com/jsp/jstl/fmt" version="2.0">
<jsp:directive.page contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" session="true" />
<jsp:output doctype-root-element="html"
doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
omit-xml-declaration="true" />

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title> Thank you - Foods R Us </title>
<meta name="viewport" content="width=device-width, initial-scale=1" />
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/
bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/
SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous" />                    <link href="css/style.css"
rel="stylesheet" type="text/css"></link>
<link href="css/style.css" rel="stylesheet" type="text/css" />

</head>
<body class="container" background="img/backdrop2.png">
<div>
<br/>
</div>
<!-- STRAT OF NAVIGATION -->
<nav class="navbar navbar-expand-lg navbar-light rounded-top "
style="background-color: #f8b38d; ">
```

```
<!-- START OF LOGO -->
<div class="navbar-brand col-md-2">
<a href="Dash.do" name="home" >
<img src="img/efoods.png"  class="img-circle" width="60" height="60"
alt="" />
</a>
</div>
<!-- SEARCH -->
<form name="searchForm" action="Search.do" method="POST" class="form-
inline my-2 my-lg-0">
<input class="form-control " type="search" style="width:375px"
name="searchInput" placeholder="Search" value="${searchInputValue}">
<button class="btn btn-outline-primary my-2 my-sm-0" type="submit"
name="searchButton" value="Search">Search</button></input>
</form>
<!-- LINKS -->
<ul class="navbar-nav ml-auto">
<!-- CATALOG -->
<li class="nav-item active">
<a class ="nav-link" href="Catalog.do" name="browse"> Browse </a>
</li>
<!-- CART -->
<li class="nav-item nav-a nav-a-2 active">
<a class ="nav-link" href="Cart.do"><small>Your,</small><br />Cart <small
class="badge badge-pill badge-danger">${cart.size()}</small></a>
</li>
<!-- SIGN IN -->
<li class="nav-item active">
<form>
<c:if test="${!sessionScope.authenticated}">
<a class ="nav-link" href="Auth.do" name="authentication">Sign In</a>
</c:if>
<c:if test="${sessionScope.authenticated}">
```

```
<a class ="nav-link" href="Account.do" name="account"><small>Hello, $
{sessionScope.customer.getName().split(" ")[0]}</small><br />Your
account</a>
</c:if>
</form>
</li>
</ul>
</nav>



<div name="mainDiv" class="text-center">
<img src="img/efood.jpg" />
<div class="col-md-12"
style="background-color: #fffdfc; border-radius: 25px;">
<br />

<div id="thankYouDiv">
<h1>Thank you for your order! :)</h1>
</div>



<div id="thankYouOptionsDiv">
<p>
View all your orders in your <a href="Account.do" name="account">
account </a> page
</p>
<p>
Your order details can be found <a
href="ViewOrder.do?orderName=${orderName}" target="_blank"
name="orderDetails"> here. </a>
</p>
```

```
</div>
<br />
</div>
</div>

</body>
</html>
</jsp:root>
```

Search.jspx

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:c="http://java.sun.com/jsp/jstl/core"
xmlns:fmt="http://java.sun.com/jsp/jstl/fmt" version="2.0">
<jsp:directive.page contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" session="true" />
<jsp:output doctype-root-element="html"
doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
omit-xml-declaration="true" />

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title> Search - Foods R Us </title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/
bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/
SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous" />
<link href="css/style.css" rel="stylesheet" type="text/css" />
</head>
<body class="container" background="img/backdrop2.png">
```

```
<div>
<br/>
</div>
<!-- STRAT OF NAVIGATION -->
<nav class="navbar navbar-expand-lg navbar-light rounded-top row"
style="background-color: #f8b39d;">
<!-- START OF LOGO -->
<div class="navbar-brand col-md-2">
<a href="Dash.do" name="home" >
<img src="img/efoods.png"  class="img-circle" width="60" height="60" />
</a>
</div>
<!-- SEARCH -->
<form name="searchForm" action="Search.do" method="POST" class="form-
inline my-2 my-lg-0">
<input class="form-control " type="search" style="width:375px"
name="searchInput" placeholder="Search" value="${searchInputValue}">
<button class="btn btn-outline my-2 my-sm-0" type="submit"
name="searchButton" value="Search">Search</button></input>
</form>
<!-- LINKS -->
<ul class="navbar-nav ml-auto">
<!-- CATALOG -->
<li class="nav-item active">
<a class ="nav-link" href="Catalog.do" name="browse"> Browse </a>
</li>
<!-- CART -->
<li class="nav-item nav-a nav-a-2 active">
<a class ="nav-link" href="Cart.do"><small>Your,</small><br />Cart <small
class="badge badge-pill badge-danger">${cart.size()}</small></a>
</li>
<!-- SIGN IN -->
<li class="nav-item active">
<form>
```

```
<c:if test="${!sessionScope.authenticated}">
<a class ="nav-link" href="Auth.do" name="authentication">Sign In</a>
</c:if>
<c:if test="${sessionScope.authenticated}">
<a class ="nav-link" href="Account.do" name="account"><small>Hello, ${sessionScope.customer.getName().split(" ")[0]} </small><br />Your account</a>
</c:if>
</form>
</li>
</ul>
</nav>

<div class="advancedSearchDiv row auto " style="background-color: #fffdfc;">

<div class="searchOptions col-lg-4" >
<br/>
<form name="searchOptionsForm" action="Search.do" method="POST">
<div class="m-2">
<input type="text" name="searchInput" placeholder="Advanced Search" value="${searchInputValue}" />
</div>
<div>
<select id="sortBy" name="sortBy" class="m-2 form-control">
<option selected="any" value="NONE">Select One:</option>
<option value="A to Z">A to Z</option>
<option value="Z to A">Z to A</option>
<option value="Price - Low to High">Price - Low to High</option>
<option value="Price - High to Low"> Price - High to Low</option>
</select>
</div>
```

```
Min cost: <input class="m-2" type="text" name="minInput" value="$
{minInputValue}" />
<br/>
Max cost: <input  class="m-1" type="text"  name="maxInput" value="$
{maxInputValue}" />
<br/>

<input class="btn btn-outline m-2" type="submit"
name="advancedSearchButton" value="Search"/>
</form>
</div>
<div class="searchResultDiv col-lg-8" >
<c:if test="${!empty result}">
<div class="row ">
<c:forEach items="${result}" var="item">
<div class="card m-4" style="width: 300px;">
<form name="addToCartForm" action="Search.do" method="POST">
<div class="m-2">
Name: ${item.getName()}
<br />
QTY: ${item.getQuantity()}
<br />
Price: CAD<fmt:formatNumber  type="currency" value="${item.getPrice()}" /
>
<br />
Quantity:
<input type="number" placeholder="1" style="float: right" value="1"
name="addQuantity" step="1" min="1"/>
<input class="btn btn-outline m-2" style="float: right" type="submit"
name="cartButton" value="Add to Cart"/>
<input type="hidden" name="hiddenItemNo" value="${item.getNumber()}"/>
<br />
ID: ${item.getNumber()}
<c:if test="${cart.containsKey(item.getNumber())}">
```

```
<p style="color: green;"> ${cart.get(item.getNumber())} of this item
exists in the cart. </p>
</c:if>
</div>
</form>
</div>
</c:forEach>
</div>
</c:if>
<c:if test="${empty result}">
<h4 style="color: red;'"> No items found in your search! </h4>
</c:if>
</div>
</div>
</body>
</html>
</jsp:root>
```

SuggestedCart.jspx

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:c="http://java.sun.com/jsp/jstl/core"
xmlns:fmt="http://java.sun.com/jsp/jstl/fmt" version="2.0">
<jsp:directive.page contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" session="true" />
<jsp:output doctype-root-element="html"
doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
omit-xml-declaration="true" />

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
<title> Cart - Foods R Us </title>
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/
bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/
SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous" />
<link href="css/style.css" rel="stylesheet" type="text/css" />
</head>
<body class="container" background="img/backdrop2.png">
<!-- STRAT OF NAVIGATION -->
<div>
<br/>
</div>
<nav class="navbar navbar-expand-lg navbar-light rounded-top"
style="background-color: #f8b39d;">
<!-- START OF LOGO -->
<div class="navbar-brand col-md-2">
<a href="Dash.do" name="home" >
<img src="img/efoods.png"  class="img-circle" width="60" height="60"
alt="" />
</a>
</div>
<!-- SEARCH -->
<form name="searchForm" action="Search.do" method="POST" class="form-
inline my-2 my-lg-0">
<input class="form-control " type="search" style="width:375px"
name="searchInput" placeholder="Search" value="${searchInputValue}">
<button class="btn btn-outline-primary my-2 my-sm-0" type="submit"
name="searchButton" value="Search">Search</button></input>
</form>
<!-- LINKS -->
<ul class="navbar-nav ml-auto">
<!-- CATALOG -->
<li class="nav-item active">
<a class ="nav-link" href="Catalog.do" name="browse"> Browse </a>
</li>
```

```
<!-- CART -->
<li class="nav-item nav-a nav-a-2 active">
<a class ="nav-link" href="Cart.do"><small>Your,</small><br />Cart <small
class="badge badge-pill badge-danger">${cart.size()}</small></a>
</li>
<!-- SIGN IN -->
<li class="nav-item active">
<form>
<c:if test="${!sessionScope.authenticated}">
<a class ="nav-link" href="Auth.do" name="authentication">Sign In</a>
</c:if>
<c:if test="${sessionScope.authenticated}">
<a class ="nav-link" href="Account.do" name="account"><small>Hello, $
{sessionScope.customer.getName().split(" ")[0]} </small><br />Your
account</a>
</c:if>
</form>
</li>
</ul>
</nav>
<div class="cartDiv rounded col-md-12" style="background-color:
#fffdfc;">
<br/>
<h2>Your Cart</h2>

<div class="cartItemTableDiv" >
<c:if test="${viewableCart.isEmpty()}">
<p> Your cart is empty. Add items <a href="Catalog.do"> here. </a> </p>
</c:if>
<c:if test="${!viewableCart.isEmpty()}">
<form name="updateCartForm" action="Cart.do" method="POST">
<table class="table">
<tr class="font-weight-bold">
```

```
<td> Name </td>
<td> Quantity </td>
<td> Cost Per Unit </td>
<td> Price </td>
<td> ID </td>
<td> Delete </td>
</tr>
<c:forEach items="${viewableCart}" var="cartItem">
<input type="hidden" name="itemId" value="${cartItem.key.getNumber()}"/>
<tr>
<td>${cartItem.key.getName()}</td>
<td>x <input type="number" min="0" step="1" value="${cartItem.value}"
name="quantityInput"/></td>
<td>CAD<fmt:formatNumber  type="currency" value="$
{cartItem.key.getPrice()}" /></td>
<td>CAD<fmt:formatNumber  type="currency" value="$
{cartItem.key.getPrice() * cartItem.value}" /></td>
<td>${cartItem.key.getNumber()}</td>
<td> Delete: <input type="checkbox" value="${cartItem.key.getNumber()}"
name="deleteCheckbox"/> </td>
</tr>
</c:forEach>
</table>
<div class="text-right">
<button class="btn btn-outline" type="submit"
name="updateCartButton">Update</button>
</div>
<c:if test="${!empty error}">
<p style="color: red;"> <strong> ${error}</strong></p>
</c:if>
</form>
</c:if>
</div>
<div name="lowerContent" class="row ">
```

```
<div name="suggestionDiv" class="col" syle="background-color: red;">
<h2>Suggested Items:</h2>
<br />
Because you bought: <br /> <c:forEach items="${suggestedList}"
var="suggestedItem"> ${suggestedItem.getName()} <br /></c:forEach>
<br />
We suggest you buy: <br />
<table class="table">
<form name="cartItemForm" action="Search.do" method="POST">
<c:forEach items="${suggestedCart}" var="suggestedItem">
<input type="hidden" name="itemId" value="${cartItem.key.getNumber()}"/>
<tr>
<td>${suggestedItem.getName()}</td>
<td>CAD<fmt:formatNumber  type="currency" value="$
{suggestedItem.getPrice()}" /></td>
<td>${suggestedItem.getNumber()}</td>
<td><input type="hidden" value="1" name="addQuantity" />
<input class="btn btn-outline m-2" type="submit" name="cartButton"
value="Add to Cart"/>
<input type="hidden" name="hiddenItemNo" value="$
{suggestedItem.getNumber()}"/>
</td>
</tr>
</c:forEach>
</form>
</table>


</div>


<div name="checkoutDiv" class="col">
<form name="checkoutForm" action="Checkout.do" method="POST">
<h2> Price Calculations: </h2>
<p name="itemsCost"> Cost of Items: CAD<fmt:formatNumber  type="currency"
value="${itemsCost}" /> </p>
```

```
<p name="hstAmount"> HST: CAD<fmt:formatNumber  type="currency" value="$
{hstAmount}" /> </p>
<p name="shippingCost"> Shipping Cost: CAD<fmt:formatNumber
type="currency" value="${shippingCost}" /> </p>
<p name="overallCost"> Total Cost: CAD<fmt:formatNumber  type="currency"
value="${shippingCost+(itemsCost+hstAmount)}" /> </p>


</form>
<div class="row">
<form name="checkoutForm" action="Checkout.do" method="POST">
<input class="btn btn-outline m-1 col" type="submit"
name="checkoutButton" value="Checkout"/>
</form>
<div class="col-5" />
<form name="checkoutForm" action="Catalog.do" method="POST">
<input class="btn btn-outline m-1 col" type="submit"
name="continueShoppingButton" value="Continue Shopping"/>
</form>
</div>
</div>
</div>
<br/>
</div>
</body>
</html>
</jsp:root>
```

ViewOrder.jspx

```
<?xml version="1.0" encoding="UTF-8" ?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:c="http://java.sun.com/jsp/jstl/core"
xmlns:fmt="http://java.sun.com/jsp/jstl/fmt" version="2.0">
<jsp:directive.page contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" session="false" />
```

```
<jsp:output doctype-root-element="html"
doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
omit-xml-declaration="true" />
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>${orderFileName}</title>
</head>
<body>
<c:set var="customer" value="${order.getCustomer()}"></c:set>
<c:set var="items" value="${order.getItems()}"></c:set>
<div id="orderInfo">
<h3>Order Info:</h3>
<p>Order id: ${order.getId()}</p>
<p>Submitted on: ${order.getSubmitted()}</p>
</div>

<hr />
<div id="customerInfo">
<h3>Customer Info:</h3>
<p>Customer Name: ${customer.getName()}</p>
<p>Customer Account: ${customer.getAccount()}</p>
</div>
<hr />
<div id="itemInfo">
<h3>Purchase Info:</h3>
<table border="1">
<tr>
<th>Item Number</th>
<th>Item Name</th>
<th>Price</th>
<th>Quantity</th>
```

```
<th>Total Price</th>
</tr>
<c:forEach var="item" items="${items}">
<tr>
<td>${item.getNumber()}</td>
<td>${item.getName()}</td>
<td>CAD<fmt:formatNumber value="${item.getPriceFormat()}"
type="currency">
</fmt:formatNumber></td>
<td>${item.getQuantity()}</td>
<td>CAD<fmt:formatNumber value="${item.getExtendedFormat()}"
type="currency"></fmt:formatNumber></td>

</tr>
</c:forEach>
</table>
</div>
<hr />
<div id="costInfo">
<h3>Cost Info:</h3>
<p>
Total: CAD<fmt:formatNumber type="currency" value="$
{order.getTotalFormat()}"></fmt:formatNumber>
</p>
<p>
Shipping: CAD<fmt:formatNumber type="currency" value="$
{order.getShippingFormat()}"></fmt:formatNumber>
</p>
<p>
HST: CAD<fmt:formatNumber type="currency" value="$
{order.getHSTFormat()}"></fmt:formatNumber>
</p>
<p>
```

```
Grand Total: CAD<fmt:formatNumber type="currency" value="$
{order.getGrandTotalFormat()}"></fmt:formatNumber>
</p>


</div>




</body>
</html>
</jsp:root>
```

Control

Account.java

```
package ctrl;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import javax.xml.bind.JAXBException;

import model.CustomerBean;
import model.Engine;

/**
 * Servlet implementation class Account
 */
@WebServlet("/Account.do")
public class Account extends HttpServlet {
	private static final long serialVersionUID = 1L;

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		HttpSession session = request.getSession();
```

```
			if ((boolean) session.getAttribute("authenticated")) {
				CustomerBean customer = (CustomerBean)
session.getAttribute("customer");
				request.setAttribute("username",
customer.getName().toString().split(" ")[0]);

				this.getServletContext().getRequestDispatcher("/
Account.jspx").forward(request, response);
			} else {
				response.sendRedirect("/eFoods/Dash.do");
			}

	}

	protected void doPost(HttpServletRequest request, HttpServletResponse response)
				throws ServletException, IOException {
			// TODO Auto-generated method stub
			doGet(request, response);
	}

}
```

Admin.java

```
package ctrl;

import java.io.IOException;
import java.util.LinkedList;
import java.util.List;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import model.Engine;

/**
 * Servlet implementation class Admin
 */
@WebServlet("/Admin.do")
public class Admin extends HttpServlet {
	private static final long serialVersionUID = 1L;
```

```
	protected void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		Engine model = Engine.getInstance();
		ServletContext context = this.getServletContext();

		List<Integer> timeBetweenAdd = (List<Integer>)
context.getAttribute("timeBetweenAdd");
		List<Integer> timeBetweenCheckout = (List<Integer>)
context.getAttribute("timeBetweenCheckout");

		try {
			int averageAdd = model.getAverageTime(timeBetweenAdd);
			request.setAttribute("averageAdd", averageAdd);
		} catch (IllegalArgumentException e) {
			request.setAttribute("averageAdd", e.getMessage());
		}

		try {
			int averageCheckout = model.getAverageTime(timeBetweenCheckout);
			request.setAttribute("averageCheckout", averageCheckout);
		} catch (IllegalArgumentException e) {
			request.setAttribute("averageCheckout", e.getMessage());
		}

		this.getServletContext().getRequestDispatcher("/
Admin.jspx").forward(request, response);

	}

	protected void doPost(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		doGet(request, response);
	}

}
```

Auth.java

```
package ctrl;

import java.io.IOException;
import java.util.Map;
import java.util.TreeMap;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import javax.xml.bind.JAXBException;

import model.CustomerBean;
import model.Engine;
import model.OrderBean;

/**
 * Servlet implementation class Auth
 */
@WebServlet("/Auth.do")
public class Auth extends HttpServlet {
	private static final long serialVersionUID = 1L;
	private static final String REDIRECT = "https://www.eecs.yorku.ca/~roumani/
servers/auth/oauth.cgi?back=http://%s:%s/eFoods/Auth.do";

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {

		HttpSession session = request.getSession();
		if (request.getParameter("name") == null && request.getParameter("user")
== null
				&& request.getParameter("hash") == null) {
			String referer = request.getHeader("referer");
			session.setAttribute("referer", referer);

			String authServer = String.format(REDIRECT,
request.getServerName(), request.getServerPort());
			response.sendRedirect(authServer);
		} else {
			Engine model = Engine.getInstance();
			CustomerBean customer = new CustomerBean();

			customer.setAccount(request.getParameter("user"));
			customer.setName(request.getParameter("name"));

			session.setAttribute("customer", customer);
			session.setAttribute("authenticated", true);

			try {
				Map<String, OrderBean> previousOrders =
model.getCustomerOrders(customer);

				if (!previousOrders.isEmpty()) {
```

```
                                TreeMap<String, OrderBean> ordersTree =
(TreeMap<String, OrderBean>) previousOrders;
                                OrderBean lastOrder =
ordersTree.lastEntry().getValue();
                                session.setAttribute("lastOrder", lastOrder);
                        }

                        session.setAttribute("previousOrders", previousOrders);

                } catch (JAXBException e) {
                        response.getWriter().write("Fatal error " +
e.getMessage());
                }

                String referer = (String) session.getAttribute("referer");
                response.sendRedirect(referer);

        }

    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }

}
```

Cart.java

```
package ctrl;

import java.io.IOException;
import java.util.Map;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import model.CustomerBean;
import model.Engine;
import model.ItemBean;
```

```
/**
 * Servlet implementation class Cart
 */
@WebServlet("/Cart.do")
public class Cart extends HttpServlet {
	private static final long serialVersionUID = 1L;

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		Engine engine = Engine.getInstance();
		HttpSession session = request.getSession();

		Map<String, Integer> cart = (Map<String, Integer>)
request.getSession().getAttribute("cart");
		Map<ItemBean, Integer> viewableCart = null;
		try {
			viewableCart = engine.makeViewableCart(cart);
			request.setAttribute("viewableCart", viewableCart);
		} catch (Exception e) {
			request.setAttribute("error", "Please enter a valid input");
		}

		if (request.getParameter("updateCartButton") != null) {
			String[] itemIds = request.getParameterValues("itemId");
			String[] itemQuantities =
request.getParameterValues("quantityInput");
			String[] deleteCheckboxes = null;
			System.out.println(itemQuantities.length + " " + itemIds.length);
			if (request.getParameterValues("deleteCheckbox") != null) {
				deleteCheckboxes =
request.getParameterValues("deleteCheckbox");
			}
			Map<String, Integer> newCart = null;
			try {
				newCart = engine.updateCart(cart, itemIds, itemQuantities,
deleteCheckboxes);
				request.getSession().setAttribute("cart", newCart);
				try {
					viewableCart = engine.makeViewableCart(newCart);
					request.setAttribute("viewableCart", viewableCart);
				} catch (Exception e) {
					request.setAttribute("error", "Please enter a valid
input");
				}
			} catch (Exception e) {
```

```
                        request.setAttribute("error", "Please enter a valid
input");
                }
            }

            if (engine.isCartEmpty(cart)) {
                request.setAttribute("itemsCost", 0.0);
                request.setAttribute("hstAmount", 0.0);
                request.setAttribute("shippingCost", 0.0);
            } else {
                double itemsCost = engine.getItemsCost(viewableCart);
                double shippingCost = engine.getShippingCost(itemsCost);
                double hstAmount = engine.getHstAmount(itemsCost, shippingCost);
                request.setAttribute("itemsCost", itemsCost);
                request.setAttribute("hstAmount", hstAmount);
                request.setAttribute("shippingCost", shippingCost);
            }

            request.setAttribute("cart", session.getAttribute("cart"));
            this.getServletContext().getRequestDispatcher("/
Cart.jspx").forward(request, response);
        }

        protected void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
            doGet(request, response);
        }

}
```

Catalog.java

```
package ctrl;

import java.io.IOException;
import java.util.List;
import java.util.Map;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import model.CategoryBean;
import model.Engine;
```

```
import model.ItemBean;

/**
 * Servlet implementation class Catalog
 */
@WebServlet("/Catalog.do")
public class Catalog extends HttpServlet {
	private static final long serialVersionUID = 1L;

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {

		// Regular instantiation before anything occurs:
		Engine engine = Engine.getInstance();
		HttpSession session = request.getSession();
		request.setAttribute("cart", session.getAttribute("cart"));
		request.setAttribute("sortBy", "NONE");

		// We get the categories that exist to populate the user page with options.
		try {
			List<CategoryBean> result = engine.getAllCategories();
			request.setAttribute("catalogList", result);
		} catch (Exception e) {
			e.printStackTrace();
		}

		// If the sort by button is clicked:
		if (request.getParameter("sortByButton") != null) {
			String sortBy = request.getParameter("sortBy");
			request.setAttribute("sortBy", sortBy);
			// if a catalog was selected sort the catalog items specifically.
			if (request.getParameter("catalogId") != null && !
request.getParameter("catalogId").equals("")) {

				String catalogId = (String) request.getParameter("catalogId");
				try {
					// Lets the user know which category we are looking at.
					request.setAttribute("selectedCatalogName",
engine.getCategory(catalogId).getName());
					request.setAttribute("catalogId", catalogId);
					List<ItemBean> itemList =
engine.getCategoryItems(catalogId, sortBy);
					request.setAttribute("itemList", itemList);
				} catch (Exception e) {
					e.printStackTrace();
				}
```

```
                } else { // If no catalog is selected, then it should sort all the items
                        try {
                                // Lets the user know which category we are looking at.
                                request.setAttribute("selectedCatalogName", "All items");
                                request.setAttribute("catalogId", null);
                                List<ItemBean> itemList = engine.getAllItems(sortBy);
                                request.setAttribute("itemList", itemList);
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }
        } else if (request.getParameter("cartButton") != null) { // If the add to cart
item is clicked
                String sortBy = request.getParameter("sortBy");
                request.setAttribute("sortBy", sortBy);
                // We similarly resort the page
                // If a catalog was selected sort the catalog items specifically.
                if (request.getParameter("catalogId") != null && !
request.getParameter("catalogId").equals("")) {

                        String catalogId = (String) request.getParameter("catalogId");
                        try {
                                // Lets the user know which category we are looking at.
                                request.setAttribute("selectedCatalogName",
engine.getCategory(catalogId).getName());
                                request.setAttribute("catalogId", catalogId);
                                List<ItemBean> itemList =
engine.getCategoryItems(catalogId, sortBy);
                                request.setAttribute("itemList", itemList);
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                } else {
                        try {
                                // Lets the user know which category we are looking at.
                                request.setAttribute("selectedCatalogName", "All items");
                                request.setAttribute("catalogId", null);
                                List<ItemBean> itemList = engine.getAllItems(sortBy);
                                request.setAttribute("itemList", itemList);
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }
                Map<String, Integer> cart = (Map<String, Integer>)
request.getSession().getAttribute("cart");
                String item = request.getParameter("hiddenItemBeanId");
```

```
                        String quantity = request.getParameter("addQuantity");
                        try {
                                Map<String, Integer> newCart = engine.addItemToCart(cart, item,
quantity);
                                request.getSession().setAttribute("cart", newCart);
                                request.setAttribute("sortBy", sortBy);
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                } else {
                        if (request.getParameter("catalogId") == null) {
                                try {
                                        List<ItemBean> itemList = engine.getAllItems();
                                        request.setAttribute("itemList", itemList);
                                        request.setAttribute("selectedCatalogName", "All items");
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                        } else {
                                String catalogId = request.getParameter("catalogId");
                                request.setAttribute("catalogId", catalogId);
                                try {
                                        List<ItemBean> itemList =
engine.getCategoryItems(catalogId);
                                        request.setAttribute("itemList", itemList);
                                        request.setAttribute("selectedCatalogName",
engine.getCategory(catalogId).getName());
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                        }
                }
                this.getServletContext().getRequestDispatcher("/Catalog.jspx").forward(request,
response);
        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
         *      response)
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
                // TODO Auto-generated method stub
                doGet(request, response);
        }
```

```
}
```

Checkout.java

```
package ctrl;

import java.io.IOException;
import java.util.Map;
import java.util.TreeMap;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import javax.xml.bind.JAXBException;

import model.CustomerBean;
import model.Engine;
import model.ItemBean;
import model.OrderBean;

@WebServlet("/Checkout.do")
public class Checkout extends HttpServlet {
	private static final long serialVersionUID = 1L;

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		HttpSession session = request.getSession();
		Engine engine = Engine.getInstance();
		request.setAttribute("cart", session.getAttribute("cart"));
		if ((boolean) session.getAttribute("authenticated")) {
			Map<String, Integer> cart = (Map<String, Integer>)
session.getAttribute("cart");
			CustomerBean customer = (CustomerBean)
session.getAttribute("customer");
			try {
				// Let's get the cart from the session's information:
				Map<ItemBean, Integer> viewableCart =
engine.makeViewableCart(cart);
				request.setAttribute("viewableCart", viewableCart);
				if (request.getParameter("checkoutButton") != null
						&& !((Map<String, Integer>)
session.getAttribute("cart")).isEmpty()) {
					OrderBean order = engine.makeOrder(viewableCart,
customer);
```

```
                                engine.checkOut(order);
                        }
                } catch (Exception e) {
                        e.printStackTrace();
                }

                this.getServletContext().getRequestDispatcher("/
Checkout.jspx").forward(request, response);
        } else {
                response.sendRedirect("Auth.do");
        }

    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
            doGet(request, response);
    }

}
```

Dash.java

```
package ctrl;

import java.io.IOException;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import model.CategoryBean;
import model.CustomerBean;
import model.Engine;

@WebServlet(name = "Controller", urlPatterns = { "/index.html", "/Dash.do" })
public class Dash extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {

        Engine engine = Engine.getInstance();
```

```
		HttpSession session = request.getSession();

		try {
			List<CategoryBean> result = engine.getAllCategories();
			request.setAttribute("catalogList", result);
			request.setAttribute("cart", session.getAttribute("cart"));
		} catch (Exception e) {
			e.printStackTrace();
		}
		request.getSession(true);
		this.getServletContext().getRequestDispatcher("/
Dash.jspx").forward(request, response);
	}

	protected void doPost(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		doGet(request, response);
	}

}
```

Order.java

```
package ctrl;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import javax.xml.bind.JAXBException;

import model.CustomerBean;
import model.Engine;
import model.OrderBean;

@WebServlet("/Order.do")
public class Order extends HttpServlet {
	private static final long serialVersionUID = 1L;

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
                throws ServletException, IOException {
            // We have to set an attribute in the checkout.java called "orderName"
and if
            // that's null, then the user
            // got the page without completing an order and we need to let them know
to go
            // awy.
            Engine model = Engine.getInstance();
            HttpSession session = request.getSession();
            CustomerBean customer = (CustomerBean) session.getAttribute("customer");
            session.setAttribute("cart", new HashMap<String, Integer>());

            try {
                Map<String, OrderBean> previousOrders =
model.getCustomerOrders(customer);

                if (!previousOrders.isEmpty()) {
                    TreeMap<String, OrderBean> ordersTree = (TreeMap<String,
OrderBean>) previousOrders;
                    OrderBean lastOrder = ordersTree.lastEntry().getValue();
                    session.setAttribute("lastOrder", lastOrder);
                    request.setAttribute("orderName",
ordersTree.lastEntry().getKey());
                }

                session.setAttribute("previousOrders", previousOrders);

            } catch (JAXBException e) {
                response.getWriter().write("Fatal error " + e.getMessage());
            }
            this.getServletContext().getRequestDispatcher("/
Order.jspx").forward(request, response);
        }

        protected void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
            doGet(request, response);
        }

}
```

Search.java

```
package ctrl;

import java.io.IOException;
import java.util.List;
```

```
import java.util.Map;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import model.Engine;
import model.ItemBean;

@WebServlet("/Search.do")
public class Search extends HttpServlet {
	private static final long serialVersionUID = 1L;

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {

		Engine engine = Engine.getInstance();
		String searchInputValue = request.getParameter("searchInput");
		HttpSession session = request.getSession();
		request.setAttribute("cart", session.getAttribute("cart"));

		if (request.getParameter("searchButton") != null) {
			if (!searchInputValue.isEmpty()) {
				try {
					List<ItemBean> result =
engine.doSearch(searchInputValue);
					request.setAttribute("result",  result);
					request.setAttribute("searchInputValue",
searchInputValue);
				} catch (Exception e) {
					System.out.println(e.getMessage());
				}
			}
			this.getServletContext().getRequestDispatcher("/
Search.jspx").forward(request, response);
		} else if (request.getParameter("advancedSearchButton") != null) {

			String min = request.getParameter("minInput");
			String max = request.getParameter("maxInput");
			String sort = request.getParameter("sortBy");
			if (!searchInputValue.isEmpty()) {
				try {
```

```
                                List<ItemBean> result =
engine.doAdvanceSearch(searchInputValue, min, max, sort);
                                request.setAttribute("searchInputValue",
searchInputValue);
                                request.setAttribute("result",  result);
                        } catch (Exception e) {
                                System.out.println(e.getMessage());
                        }
                }
                this.getServletContext().getRequestDispatcher("/
Search.jspx").forward(request, response);
        } else if (request.getParameter("cartButton") != null) { // If the add to
cart item is clicked
                // We similarly resort the page

                Map<String, Integer> cart = (Map<String, Integer>)
request.getSession().getAttribute("cart");
                String item = request.getParameter("hiddenItemNo");
                String quantity = request.getParameter("addQuantity");
                try {
                        Map<String, Integer> newCart = engine.addItemToCart(cart,
item, quantity);
                        request.getSession().setAttribute("cart", newCart);
                } catch (Exception e) {
                        System.out.println(e.getMessage());
                }
                response.sendRedirect("Cart.do");
        }
        //this.getServletContext().getRequestDispatcher("/
Search.jspx").forward(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
        doGet(request, response);
    }

}
```

ViewOrder.java

```
package ctrl;

import java.io.IOException;
import java.util.Map;

import javax.servlet.ServletException;
```

```
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import model.CustomerBean;
import model.Engine;
import model.OrderBean;

/**
 * Servlet implementation class ViewOrder
 */
@WebServlet("/ViewOrder.do")
public class ViewOrder extends HttpServlet {
	private static final long serialVersionUID = 1L;

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		Engine model = Engine.getInstance();
		HttpSession session = request.getSession();
		String orderFileName = request.getParameter("orderName");
		CustomerBean customer = (CustomerBean) session.getAttribute("customer");

		boolean authenticated = (boolean) session.getAttribute("authenticated");
		boolean rightCustomer = model.isCustomerOrder(orderFileName,
customer.getAccount());

		if (orderFileName != null && authenticated && rightCustomer) {

			Map<String, OrderBean> orders = (Map<String, OrderBean>)
session.getAttribute("previousOrders");
			OrderBean order = orders.get(request.getParameter("orderName"));

			if (order != null) {
				request.setAttribute("order", order);
				request.setAttribute("orderFileName", orderFileName);
				this.getServletContext().getRequestDispatcher("/
ViewOrder.jspx").forward(request, response);
			}

		}

		response.getWriter().write("You are not authorized to view this page");

	}
```

```
	protected void doPost(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		// TODO Auto-generated method stub
		doGet(request, response);
	}

}
```

Model

CategoryBean.java

```
package model;

/**
 * Class to represent the category table from the database. Picture is stored as
 * a Base64 string that is rendered in the jspx as a viewable picture.
 *
 */
public class CategoryBean {

	private String description;
	private String name;
	private String picture;

	private int id; // SQL Key

	public CategoryBean() {
	}

	@Override
	public String toString() {
		return "CategoryBean [description=" + description + ", name=" + name + ",
id=" + id + "]";
	}
```

```
	public String getDescription() {
		return description;
	}

	public void setDescription(String description) {
		this.description = description;
	}

	public String getName() {
		return name;
	}

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

	public int getId() {
		return id;
	}

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

	public String getPicture() {
		return picture;
	}

	public void setPicture(String picture) {
		this.picture = picture;
	}
}
```

CategoryDAO.java

```
package model;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.Base64;
```

```
import java.util.Base64.Encoder;

/**
 * Data access layer for the category table.
 *
 */
public class CategoryDAO {

	// Derby information
	public static final String DERBY_DRIVER = "org.apache.derby.jdbc.ClientDriver";
	public static final String SET_SCHEMA = "set schema roumani";
	public static final String DB_URL = "jdbc:derby://localhost:64413/
EECS;user=student;password=secret";

	// Query strings to get categories
	public static final String ALL_CATEGORIES_QUERY = "SELECT * FROM CATEGORY";
	public static final String SINGLE_CATEGORY_QUERY = "SELECT * FROM CATEGORY
WHERE ID = ?";

	private Connection con;

	// Encodes the picture bytes into a Base64 String.
	private Encoder picEncoder;

	/**
	 * Constructs the DAO, initializing the database driver and creating the
	 * encoder.
	 */
	public CategoryDAO() {

		try {
			Class.forName(DERBY_DRIVER).newInstance();
			con = DriverManager.getConnection(DB_URL);
		} catch (Exception e) {
			System.err.println(e.getMessage());
			e.printStackTrace();
		}

		picEncoder = Base64.getEncoder();
	}

	// Sets the schema before each database call.
	private void setSchema() throws SQLException {
		Statement setRoumani;
		setRoumani = con.createStatement();
		setRoumani.executeUpdate(SET_SCHEMA);
```

```
	}

	/**
	 * Gets all the categories in the database and returns them in a list.
	 *
	 * @return list containing all categories
	 * @throws Exception
	 *             if an SQL occurs or if there are no categories.
	 */
	public List<CategoryBean> getAllCategories() throws Exception {
		PreparedStatement searchStatement;

		ResultSet categoryResults;
		List<CategoryBean> categoryList;

		setSchema();

		searchStatement = con.prepareStatement(ALL_CATEGORIES_QUERY);

		categoryResults = searchStatement.executeQuery();

		categoryList = makeCategoryList(categoryResults);
		return categoryList;
	}

	/**
	 * Returns the category that has the same key as the catId.
	 *
	 * @param catId
	 *            a valid category id number.
	 * @return a CategoryBean matching the category id.
	 * @throws SQLException
	 *             if there is a database error or if there are no categories with
	 *             that number.
	 */
	public CategoryBean getCategory(int catId) throws SQLException {
		PreparedStatement searchStatement;
		ResultSet categoryResults;

		setSchema();

		searchStatement = con.prepareStatement(SINGLE_CATEGORY_QUERY);
		searchStatement.setInt(1, catId);

		categoryResults = searchStatement.executeQuery();
		categoryResults.next();
```

```
		CategoryBean category = setCategoryBean(categoryResults);
		return category;
	}

	// Loops through the result and makes the list.
	private List<CategoryBean> makeCategoryList(ResultSet r) throws SQLException {
		List<CategoryBean> categoryList = new ArrayList<>();

		while (r.next()) {
			CategoryBean category = setCategoryBean(r);

			categoryList.add(category);
		}
		return categoryList;
	}

	// Populates the category bean with the current pointer of the result set.
	private CategoryBean setCategoryBean(ResultSet r) throws SQLException {
		CategoryBean category = new CategoryBean();

		category.setDescription(r.getString("DESCRIPTION"));
		category.setName(r.getString("NAME"));
		category.setId(r.getInt("ID"));
		category.setPicture(getCategoryPicture(r.getBytes("PICTURE")));

		return category;
	}

	// Encodes the picture byte array as a Base64 string.
	private String getCategoryPicture(byte[] picture) {
		byte[] encoded = picEncoder.encode(picture);

		String encodedString = new String(encoded);

		return encodedString;
	}

}
```

Engine.java

```
package model;

import java.io.File;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
```

```
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.TreeMap;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;

/**
 * Back-end logic singleton for the webstore app. Returns data from the DAO and
 * handles the business logic of the eFoods application.
 *
 */
public class Engine {

	private static Engine instance = null;
	private ItemDAO itemDao;
	private CategoryDAO catDao;

	private long fileCount;
	private static final String PO_PATH = System.getProperty("user.home") + "/PO/";
	private static final String IN_PO = PO_PATH + "inPO/";
	private static final String OUT_PO = PO_PATH + "outPO";

	private JAXBContext orderContext;
	private Marshaller orderMarshaller;
	private Unmarshaller orderUnMarshaller;

	private static final double SHIPPING_FEE = 5.0;
	private static final double HST = 0.13;
	private static final double FREE_SHIPPING = 100.0;
	private static final String itemMatcher = "([0-9]{4}[a-z|A-Z][0-9]{3})";

	// Initializes DAO's, the PO folders required on disk, and the marshallers.
	private Engine() {
		this.itemDao = new ItemDAO();
		this.catDao = new CategoryDAO();
		initPoFolder();

		try {
			this.orderContext = JAXBContext.newInstance(OrderBean.class);
			this.orderMarshaller = orderContext.createMarshaller();
			this.orderUnMarshaller = orderContext.createUnmarshaller();
```

```
                this.orderMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
Boolean.TRUE);
            } catch (JAXBException e) {
                System.err.println("Fatal error " + e.getMessage());
            }

        }

        // Creates the required PO directories and initializes the file count field.
        private void initPoFolder() {
            File poDir = new File(PO_PATH);
            File inDir = new File(IN_PO);
            File outDir = new File(OUT_PO);

            poDir.mkdirs();
            inDir.mkdir();
            outDir.mkdir();

            this.fileCount = inDir.listFiles().length + outDir.listFiles().length;
        }

        /**
         * Gets the instance of the singleton class.
         *
         * @return the single Engine object.
         */
        public static Engine getInstance() {
            if (instance == null) {
                instance = new Engine();
            }

            return instance;
        }

        /**
         * Returns a single itemBean matched with the unique item code.
         *
         * @param itemId
         *             a valid 8 character ID.
         * @return an ItemBean corresponding to the entered item id.
         * @throws Exception
         *              If the itemId does not correspond to any item, or if there is a
         *              backend exception.
         */
        public ItemBean getItem(String itemId) throws Exception {
            return itemDao.getItem(itemId);
```

```
	}

	/**
	 * Returns every available item as a list.
	 *
	 * @return a list containing every available item.
	 * @throws Exception
	 *             if an SQL exception is thrown.
	 */
	public List<ItemBean> getAllItems() throws Exception {
		return itemDao.getAllItems();
	}

	/**
	 * Returns every available item as a list, sorted by the input.
	 *
	 * @param sortBy
	 *            an input from the select tag in html
	 * @return a list containing every available item sorted by what the user
wants.
	 * @throws Exceptionif
	 *             an SQL exception is thrown.
	 */
	public List<ItemBean> getAllItems(String sortBy) throws Exception {
		return itemDao.getAllItems(sortBy);
	}

	/**
	 * Returns a single categorybean containing all the information about that
	 * category.
	 *
	 * @param catId
	 *            a valid category id
	 * @return
	 * @throws Exception
	 *             if an SQL exception is thrown.
	 */
	public CategoryBean getCategory(String catId) throws Exception {
		return catDao.getCategory(Integer.parseInt(catId));
	}

	/**
	 * Returns a list of every category.
	 *
	 * @return a list of every category.
	 * @throws Exception
```

```
	 *             if an SQL exception is thrown.
	 */
	public List<CategoryBean> getAllCategories() throws Exception {
		return catDao.getAllCategories();
	}

	/**
	 * Creates a list of items that are in the entered category.
	 *
	 * @param catId
	 *            a valid category Id.
	 * @return a list of items in the entered category ID.
	 * @throws Exception
	 *             if an SQL exception is thrown.
	 */
	public List<ItemBean> getCategoryItems(String catId) throws Exception {
		CategoryBean category = getCategory(catId);
		List<ItemBean> items = itemDao.getAllItems();
		List<ItemBean> categoryItems = new ArrayList<>();

		for (ItemBean item : items) {
			if (category.getId() == item.getCatId()) {
				categoryItems.add(item);
			}
		}

		return categoryItems;
	}

	/**
	 * Retrieves all items with a given category ID and that are sorted by the
given
	 * parameter.
	 *
	 * @param catId
	 *            a valid category Id.
	 * @param sortBy
	 *            an input from the select tag in html.
	 * @return a list of items in the category, sorted.
	 * @throws Exception
	 *             if an SQL exception is thrown.
	 */
	public List<ItemBean> getCategoryItems(String catId, String sortBy) throws
Exception {
		CategoryBean category = getCategory(catId);
		List<ItemBean> items = itemDao.getAllItems(sortBy);
```

```
        List<ItemBean> categoryItems = new ArrayList<>();

        for (ItemBean item : items) {
                if (category.getId() == item.getCatId()) {
                        categoryItems.add(item);
                }
        }

        return categoryItems;
}

/**
 * Searches for items that match an input, and then returns the list.
 *
 * @param searchInputValue
 *             a string to search from.
 * @return a list containing itemBeans that contain a substring of the search
 *         string.
 * @throws Exception
 *              if there is an SQL error or if the list returned is empty.
 */
public List<ItemBean> doSearch(String searchInputValue) throws Exception {
        List<ItemBean> result = new ArrayList<>();
        if (searchInputValue.isEmpty()) {
                throw new IllegalArgumentException("");
        }
        if (searchInputValue.matches(itemMatcher)) {
                result.add(getItem(searchInputValue));
        } else {
                result = itemDao.search(searchInputValue);

                if (result.isEmpty()) {
                        throw new Exception("No results found.");
                }
        }
        return result;
}

/**
 * Search for an item or items with a given min price, max price and sorting
 * criteria
 *
 * @param searchInputValue
 *             a string to search from.
 * @param minCost
 *             the minimum cost of an item.
```

```
        * @param maxCost
        *            the maximum cost of an item.
        * @param sortBy
        *            an input from the select tag in html.
        * @return a list of items that match the entered parameters.
        * @throws Exception
        *             if an SQL exception is thrown.
        */
       public List<ItemBean> doAdvanceSearch(String searchInputValue, String minCost,
String maxCost, String sortBy)
                     throws Exception {
              List<ItemBean> result = new ArrayList<>();
              if (searchInputValue.isEmpty()) {
                     throw new IllegalArgumentException("Search query is empty.");
              }
              if (searchInputValue.matches(itemMatcher)) {
                     result =
itemDao.advanceSearch((getItem(searchInputValue).getName()), minCost, maxCost,
sortBy);
              } else {
                     result = itemDao.advanceSearch(searchInputValue, minCost, maxCost,
sortBy);
              }
              if (result.isEmpty()) {
                     throw new Exception("No results returned.");
              }

              return result;
       }

       /**
        * This method adds an item to the shopping cart within the session. If the
item
        * exists, it appends the original amount with the new quantity. If the item
        * doesn't exist, it creates the item with the new quantity.
        *
        * @param cart
        *            is the cart within the session.
        * @param item
        *            is the item to add or append.
        * @param quantity
        *            is the amount of the item to be added or appended by.
        * @return the Map of the cart after alterations (addition).
        * @throws Exception
        */
```

```
	public Map<String, Integer> addItemToCart(Map<String, Integer> cart, String
itemNo, String quantity)
			throws Exception {

		int quantityInt = Integer.parseInt(quantity);

		if (cart.containsKey(itemNo)) {
			cart.put(itemNo, cart.get(itemNo) + quantityInt);

		} else {
			cart.put(itemNo, quantityInt);
		}

		return cart;

	}

	/**
	 * This method removes all of an item from the cart within the session. If it
	 * does not exist, it throws an exception stating so.
	 *
	 * @param cart
	 *            is the cart within the session.
	 * @param item
	 *            is the item to be removed.
	 * @return the Map of the cart after alterations (removal).
	 */
	public Map<String, Integer> removeItemFromCart(Map<String, Integer> cart,
ItemBean item) {
		if (cart.containsKey(item.getNumber())) {
			cart.remove(item.getNumber());
		} else {
			throw new IllegalArgumentException("That item is not in the
cart!");
		}
		return cart;
	}

	/**
	 * This method is in support of the view. It creates a cart that is viewable as
	 * it contains information such as the price, name, etc. of the item as opposed
	 * to the cart that is stored in the session that only contains IDs. It gets
the
	 * rest of the information using the item ID string by calling the getItem
	 * method in this Engine.
	 *
```

```
        * @param cart
        *            is the cart within the session.
        * @return is a Map that is viewable since it has the entire ItemBean along
with
        *         the Integer quantity.
        * @throws Exception
        *             is thrown if there is an issue getting the item with the ItemNo
        *             id.
        */
       public Map<ItemBean, Integer> makeViewableCart(Map<String, Integer> cart)
throws Exception {

              Map<ItemBean, Integer> viewableCart = new LinkedHashMap<ItemBean,
Integer>();

              for (String s : cart.keySet()) {
                     viewableCart.put(this.getItem(s), cart.get(s));
              }

              return viewableCart;
       }

       /**
        * Creates an OrderBean from the viewableCart and a customerBean. The OrderBean
        * contains the customerBean and a list of ItemBeans where the quantity and
        * total price (extended) are set. The orderBean also contains shipping, HST,
        * total, and grand total pricing easily accessible.
        *
        *
        * @param viewableCart
        *            a non empty viewableCart.
        * @param customer
        *            a non-empty customerBean
        * @return
        * @throws Exception
        */
       public OrderBean makeOrder(Map<ItemBean, Integer> viewableCart, CustomerBean
customer) throws Exception {
              OrderBean order = new OrderBean();
              List<ItemBean> itemList = new ArrayList<>();
              double hst, total, grandTotal, shipping;

              total = 0.0;
              for (ItemBean item : viewableCart.keySet()) {
                     item.setQuantity(viewableCart.get(item));
                     item.setExtended(item.getQuantity() * item.getPrice());
```

```
                itemList.add(item);
                total += item.getExtended();
            }

            shipping = getShippingCost(total);
            hst = getHstAmount(total, shipping);
            grandTotal = total + hst + shipping;

            order.setItems(itemList);
            order.setSubmitted(getDate());
            order.setCustomer(customer);

            order.setTotal(total);
            order.setHST(hst);
            order.setShipping(shipping);
            order.setGrandTotal(grandTotal);

            return order;

        }

        /**
         * Gives the current date of the server as "yyyy-mm-dd"
         *
         * @return the date formatted as "yyyy-mm-dd"
         */
        private String getDate() {
            LocalDate currTime = LocalDate.now();
            DateTimeFormatter timeFormat = DateTimeFormatter.ofPattern("yyyy-MM-dd");

            String formattedTime = currTime.format(timeFormat);
            return formattedTime;
        }

        /**
         * Turns an orderBean into an XML file on disk (A recieved order). The
OrderBean
         * should be removed from the session and cart emptied after calling this
         * method.
         *
         * @param order
         *            a populated orderBean.
         * @throws Exception
         */
        public void checkOut(OrderBean order) throws Exception {
            String fileCountString = makeOrderId();
```

```
            order.setId(Integer.parseInt(fileCountString));
            String poName = "po" + order.getCustomer().getAccount() + "_" +
fileCountString + ".xml";
            File newPo = new File(IN_PO + poName);

            newPo.createNewFile();
            orderMarshaller.marshal(order, newPo);
      }

      /**
       * Creates a 2+ digit orderId for the orderBean. Used in the filename, and
       * inside the P XML.
       *
       * @return a string of the proper orderId.
       */
      private String makeOrderId() {
            String fileCountString;
            if (++fileCount < 10) {
                  fileCountString = "0" + fileCount;
            } else {
                  fileCountString = Long.toString(fileCount);
            }

            return fileCountString;
      }

      /**
       * Generates a list of customer orders based on the CustomerBean.
       *
       * @param customer
       *            a populated customerBean.
       * @return A List of orders the customer made, may be empty if the customer has
       *         made no orders.
       * @throws JAXBException
       * @throws Exception
       */
      public Map<String, OrderBean> getCustomerOrders(CustomerBean customer) throws
JAXBException {
            Map<String, OrderBean> customerOrders = new TreeMap<>();
            File inPODir[] = new File(IN_PO).listFiles();
            File outPODir[] = new File(OUT_PO).listFiles();

            for (File file : inPODir) {
                  if (isCustomerOrder(file.getName(), customer.getAccount())) {
```

```
                        OrderBean customerOrder = (OrderBean)
orderUnMarshaller.unmarshal(file);
                        customerOrders.put(file.getName(), customerOrder);
                  }
            }

            for (File file : outPODir) {
                  if (isCustomerOrder(file.getName(), customer.getAccount())) {
                        OrderBean customerOrder = (OrderBean)
orderUnMarshaller.unmarshal(file);
                        customerOrders.put(file.getName(), customerOrder);
                  }
            }

            return customerOrders;
      }

      /**
       * Checks if the customer created the order, if they did not they are not
       * allowed to view the order and the method returns false.
       *
       * @param fileName
       *            the name of the xml file the user wishes to access.
       * @param accountName
       *            the users account name in the customer session.
       * @return true if the user name matches the regex, false otherwise.
       */
      public boolean isCustomerOrder(String fileName, String accountName) {
            if (fileName.matches("po" + accountName + "_\\d+.xml")) {
                  return true;
            }

            return false;
      }

      /**
       * update the cart in session with the requested parameters which returns a map
       * of itemIds and the quantities of those items.
       *
       * @param cart
       *            the users session cart.
       * @param itemIds
       *            the ids of the items to change.
       * @param itemQuantities
       *            the new quantities for each item.
       * @return the updated cart with the new quantities.
```

```
	 */
	public Map<String, Integer> updateCart(Map<String, Integer> cart, String[]
itemIds, String[] itemQuantities,
			String[] deleteCheckboxes) throws Exception {

		for (int i = 0; i < itemIds.length; i++) {
			if (0 == Integer.parseInt(itemQuantities[i])) {
				cart.remove(itemIds[i]);
			} else if (cart.get(itemIds[i]) !=
Integer.parseInt(itemQuantities[i])) {
				cart.put(itemIds[i], Integer.parseInt(itemQuantities[i]));
			}
		}

		if (deleteCheckboxes != null) {
			for (String s : deleteCheckboxes) {
				if (cart.containsKey(s)) {
					cart.remove(s);
				}
			}
		}

		return cart;
	}

	/**
	 * Checks if the session's cart is empty.
	 *
	 * @param cart
	 *            a session cart.
	 * @return true if the cart is empty, false if not.
	 */
	public boolean isCartEmpty(Map<String, Integer> cart) {
		return cart.isEmpty();
	}

	/**
	 * Returns the cost of all items, cost and HST.
	 *
	 * @param cart
	 *            a viewable cart.
	 * @return the total cost of the items in the cart.
	 */
	public double getItemsCost(Map<ItemBean, Integer> cart) {
		double itemsCost = 0;
		for (ItemBean i : cart.keySet()) {
```

```
                itemsCost = itemsCost + i.getPrice() * cart.get(i);
        }
        return itemsCost;
}

/**
 * Determines the shipping cost for the orders. If the carts total is greater
 * than 100, shipping is free, if not shipping is equal to the shipping fee.
 *
 * @param itemscost
 *            the total cost of the items in the cart.
 * @return the shipping cost of the order.
 */
public double getShippingCost(double itemsCost) {
        if (itemsCost >= FREE_SHIPPING) {
                return 0.0;
        } else {
                return SHIPPING_FEE;
        }
}

/**
 * Calculates the hst amount of the items in the cart plus the shipping.
 *
 * @param itemsCost
 *            the total cost of the items in the cart.
 * @param shippingCost
 *            the shipping cost of the order.
 * @return the hst amount of the items in the cart added to shipping.
 */
public double getHstAmount(double itemsCost, double shippingCost) {
        double hstAmount = (itemsCost + shippingCost) * HST;
        return hstAmount;
}

/**
 * Adds up the value of every integer in the list and then divides it by the
 * list size. Used in analytics. Throws {@link IllegalArgumentException} if the
 * list is empty.
 *
 * @param analyticList
 *            one of the lists in the context used to track timing.
 * @return the average value of the list.
 */
public int getAverageTime(List<Integer> analyticList) {
```

```
			if (analyticList.size() == 0) {
				throw new IllegalArgumentException("No users have performed the
required action.");
			}

			int totalTime = 0;
			for (Integer time : analyticList) {
				totalTime += time;
			}

			totalTime /= analyticList.size();

			return totalTime;
		}

}
```

ItemBean.java

```
package model;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlTransient;
import javax.xml.bind.annotation.XmlType;

/**
 * Represents an ItemBean both for use in the program, and as an XML element.
 * Price and Extended price fields are both set as a formatted string, and as a
 * double. This is so both calculations can be done on them easily as a double,
 * but they will be properly formatted in XML.
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(propOrder = { "number", "name", "priceFormat", "quantity",
"extendedFormat" })
public class ItemBean {

	@XmlAttribute
	private String number; // 8-digit product code. Key
	private String name; // NAME
```

```
	private int quantity; // QTY

	@XmlTransient
	private double price; // PRICE
	@XmlTransient
	private double extended; // Total price, equal to quantity * price
	@XmlTransient
	private String unit; // UNIT quantity per unit
	@XmlTransient
	private int catId; // CATID category Id

	@XmlElement(name = "price")
	private String priceFormat;
	@XmlElement(name = "extended")
	private String extendedFormat;

	public ItemBean() {
		super();
	}

	public String getUnit() {
		return unit;
	}

	public void setUnit(String unit) {
		this.unit = unit;
	}

	public double getExtended() {
		return extended;
	}

	public String getExtendedFormat() {
		return this.extendedFormat;
	}

	public void setExtended(double extended) {
		this.extended = extended;
		this.extendedFormat = String.format("%.2f", this.extended);
	}

	public int getCatId() {
		return catId;
	}

	public void setCatId(int catId) {
```

```
		this.catId = catId;
	}

	public int getQuantity() {
		return quantity;
	}

	public void setQuantity(int quantity) {
		this.quantity = quantity;
	}

	public double getPrice() {
		return price;
	}

	public String getPriceFormat() {
		return this.priceFormat;
	}

	public void setPrice(double price) {
		this.price = price;
		this.priceFormat = String.format("%.2f", this.price);
	}

	public String getName() {
		return name;
	}

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

	public String getNumber() {
		return number;
	}

	public void setNumber(String number) {
		this.number = number;
	}

	@Override
	public String toString() {
		return "itemBean [unit=" + unit + ", costPrice=" + extended + ", catId="
+ catId + ", quantity=" + quantity
				+ ", price=" + price + ", name=" + name + ", number=" +
number + "]";
```

```
	}

}
```

ItemDAO.java

```
package model;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

/**
 * Data access layer for the item table in the database. Handles searching and
 * retrieving functionality. Should be called by the engine
 *
 */
public class ItemDAO {

	public static final String DERBY_DRIVER = "org.apache.derby.jdbc.ClientDriver";
	public static final String DB_URL = "jdbc:derby://localhost:64413/
EECS;user=student;password=secret";
	public static final String SET_SCHEMA = "set schema roumani";

	// Queries
	public static final String SEARCH_QUERY = "SELECT * FROM ITEM WHERE LOWER(NAME)
LIKE LOWER(?)";
	public static final String ADVANCE_QUERY = SEARCH_QUERY + " AND PRICE BETWEEN ?
AND ?";
	public static final String GET_ITEM_QUERY = "SELECT * FROM ITEM WHERE NUMBER
= ?";

	// This helps prevent SQL injection attacks on the ORDER BY statement.
	public static final String[] SORT_OPTIONS = { "NUMBER", "PRICE ASC", "PRICE
DESC", "NAME ASC", "NAME DESC" };
	public static final String[] USER_SORT_INPUT = { "NONE", "Price - Low to High",
"Price - High to Low", "A to Z",
			"Z to A" };
	private HashMap<String, String> orderMap;

	private final double MAX_RANGE_VALUE = 1000000.00;
```

```
private Connection con;

/**
 * Constructs an ItemDao, initializing the driver and constructing the orderMap
 * used to prevent SQL injections when the user wants to order their sort.
 */
public ItemDAO() {
	this.orderMap = new HashMap<>();

	try {
		Class.forName(DERBY_DRIVER).newInstance();
		con = DriverManager.getConnection(DB_URL);
	} catch (Exception e) {
		System.err.println(e.getMessage());
		e.printStackTrace();
	}

	for (int i = 0; i < SORT_OPTIONS.length; i++) {
		orderMap.put(USER_SORT_INPUT[i], SORT_OPTIONS[i]);
	}
}

// Sets schema for every database call.
private void setSchema() throws SQLException {
	Statement setRoumani;
	setRoumani = con.createStatement();
	setRoumani.executeUpdate(SET_SCHEMA);
}

/**
 * Queries for items that contain the search query.
 *
 * @param searchInputValue
 * @return a list of itembeans that match.
 * @throws Exception
 */
public List<ItemBean> search(String searchInputValue) throws Exception {
	PreparedStatement searchStatement;

	ResultSet itemResults;
	List<ItemBean> itemList;

	setSchema();

	searchStatement = con.prepareStatement(SEARCH_QUERY);
```

```
			searchStatement.setString(1, "%" + searchInputValue + "%");

			itemResults = searchStatement.executeQuery();

			itemList = makeItemList(itemResults);
			return itemList;
		}

		/**
		 * Does an advanced search that allows for more constraints on what is
returned.
		 *
		 * @param searchInputValue
		 * @param minCost
		 * @param maxCost
		 * @param sortBy
		 * @return a list of items that match the entered arguments.
		 * @throws Exception
		 */
		public List<ItemBean> advanceSearch(String searchInputValue, String minCost,
String maxCost, String sortBy)
				throws Exception {
			PreparedStatement searchStatement;

			ResultSet itemResults;
			List<ItemBean> itemList;

			setSchema();

			if (sortBy.isEmpty()) {
				searchStatement = con.prepareStatement(ADVANCE_QUERY);
			} else {
				searchStatement = con.prepareStatement(ADVANCE_QUERY + " ORDER BY
" + getSort(sortBy));
			}

			if (minCost.isEmpty()) {
				searchStatement.setDouble(2, 0.0);
			} else {
				searchStatement.setDouble(2, Double.parseDouble(minCost));
			}

			if (maxCost.isEmpty()) {
				searchStatement.setDouble(3, MAX_RANGE_VALUE);
			} else {
				searchStatement.setDouble(3, Double.parseDouble(maxCost));
```

```
        }

        searchStatement.setString(1, "%" + searchInputValue + "%");
        itemResults = searchStatement.executeQuery();
        itemList = makeItemList(itemResults);
        return itemList;

    }

    /**
     * Retrieves all items from the database.
     *
     * @return a list of all items.
     * @throws Exception
     */
    public List<ItemBean> getAllItems() throws Exception {
        PreparedStatement searchStatement;

        ResultSet itemResults;
        List<ItemBean> itemList;

        setSchema();

        searchStatement = con.prepareStatement(SEARCH_QUERY);
        searchStatement.setString(1, "%");

        itemResults = searchStatement.executeQuery();
        itemList = makeItemList(itemResults);

        return itemList;
    }

    /**
     * Gets all items in the database in the order of the entered argument.
     *
     * @param sortBy
     *            a select option from html.
     * @return all items sorted.
     * @throws Exception
     */
    public List<ItemBean> getAllItems(String sortBy) throws Exception {
        PreparedStatement searchStatement;

        ResultSet itemResults;
        List<ItemBean> itemList;
```

```
			setSchema();

			searchStatement = con.prepareStatement(SEARCH_QUERY + " ORDER BY " +
getSort(sortBy));
			searchStatement.setString(1, "%");

			itemResults = searchStatement.executeQuery();
			itemList = makeItemList(itemResults);

			return itemList;
		}

		/**
		 * Gets a single item from the database as an ItemBean, based on it's unique
		 * number.
		 *
		 * @param itemId
		 *            a valid item number.
		 * @return a single ItemBean corresponding to the item retrieved.
		 * @throws Exception
		 */
		public ItemBean getItem(String itemId) throws Exception {
			PreparedStatement searchStatement;

			ResultSet itemResults;
			ItemBean item = new ItemBean();

			setSchema();

			searchStatement = con.prepareStatement(GET_ITEM_QUERY);
			searchStatement.setString(1, itemId);

			itemResults = searchStatement.executeQuery();

			if (itemResults.next()) {
				item = setItemBean(itemResults);
			} else {
				throw new IllegalArgumentException(itemId + " is not a valid item
number.");
			}

			return item;
		}

		// Loops through the ResultSet and populates the list of ItemBeans
		private List<ItemBean> makeItemList(ResultSet r) throws SQLException {
```

```
		List<ItemBean> itemList = new ArrayList<>();

		while (r.next()) {
			ItemBean item = setItemBean(r);
			itemList.add(item);
		}

		return itemList;
	}

	// Sets each attribute on the ItemBean from the database values.
	private ItemBean setItemBean(ResultSet r) throws SQLException {
		ItemBean item = new ItemBean();

		item.setUnit(r.getString("UNIT"));
		item.setCatId(r.getInt("CATID"));
		item.setPrice(r.getDouble("PRICE"));
		item.setName(r.getString("NAME"));
		item.setNumber(r.getString("NUMBER"));

		return item;
	}

	/**
	 * Checks the input string against its value in a map and returns the mapped
	 * value. If there is no matching mapped value, an exception is thrown. This
	 * could indicate the user replaced the value of the options.
	 *
	 * @param order
	 *            input string from the user from a select element
	 * @return the sort parameter.
	 */
	private String getSort(String order) {

		String sanitizedOrder = orderMap.get(order);

		if (sanitizedOrder != null)
			return orderMap.get(order);
		else
			throw new IllegalArgumentException("Attempt to inject SQL
Detected.");

	}

}
```

OrderBean.java

```
package model;

import java.util.List;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlElementWrapper;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;
import javax.xml.bind.annotation.XmlType;

/**
 * Bean to represent an order that can be translated to and from XML.
 *
 */
@XmlRootElement(name = "order")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(propOrder = { "id", "submitted", "customer", "items", "totalFormat",
"shippingFormat", "HSTFormat",
             "grandTotalFormat" })
public class OrderBean {
       @XmlAttribute
       private int id;
       @XmlAttribute
       private String submitted;

       private CustomerBean customer;

       @XmlElementWrapper
       @XmlElement(name = "item")
       private List<ItemBean> items;

       @XmlTransient
       private double total;
       @XmlTransient
       private double shipping;
       @XmlTransient
       private double HST;
       @XmlTransient
       private double grandTotal;
```

```
@XmlElement(name = "total")
private String totalFormat;
@XmlElement(name = "shipping")
private String shippingFormat;
@XmlElement(name = "HST")
private String HSTFormat;
@XmlElement(name = "grandTotal")
private String grandTotalFormat;

public String getTotalFormat() {
	return this.totalFormat;
}

public String getShippingFormat() {
	return this.shippingFormat;
}

public String getHSTFormat() {
	return this.HSTFormat;
}

public String getGrandTotalFormat() {
	return this.grandTotalFormat;
}

public int getId() {
	return id;
}

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

public String getSubmitted() {
	return submitted;
}

public void setSubmitted(String submitted) {
	this.submitted = submitted;
}

public CustomerBean getCustomer() {
	return customer;
}

public void setCustomer(CustomerBean customer) {
```

```
		this.customer = customer;
	}

	public List<ItemBean> getItems() {
		return items;
	}

	public void setItems(List<ItemBean> items) {
		this.items = items;
	}

	public double getTotal() {
		return total;
	}

	public void setTotal(double total) {
		this.total = total;
		this.totalFormat = String.format("%.2f", total);
	}

	public double getShipping() {
		return shipping;
	}

	public void setShipping(double shipping) {
		this.shipping = shipping;
		this.shippingFormat = String.format("%.2f", shipping);
	}

	public double getGrandTotal() {
		return grandTotal;
	}

	public void setGrandTotal(double grandTotal) {
		this.grandTotal = grandTotal;
		this.grandTotalFormat = String.format("%.2f", grandTotal);
	}

	public double getHST() {
		return HST;
	}

	public void setHST(double HST) {
		this.HST = HST;
		this.HSTFormat = String.format("%.2f", HST);
	}
```

```
	@Override
	public String toString() {
		return "OrderBean [id=" + id + ", submitted=" + submitted + ", customer="
+ customer + ", items=" + items
				+ ", total=" + total + ", shipping=" + shipping + ", HST="
+ HST + ", grandTotal=" + grandTotal + "]";
	}

}
```

Ad-hoc

Reference.java

```
package adhoc;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.TreeMap;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;

import model.Engine;
import model.ItemBean;

/**
 * Servlet Filter implementation class reference
 */
@WebFilter({ "/Cart.do" })
public class Reference implements Filter {

	private static Map<String, String> crossRef = new TreeMap<>();

	public static Map<String, String> getCrossRef() {
		return crossRef;
	}
```

```
public static void setCrossRef(Map<String, String> crossRef) {
	Reference.crossRef = crossRef;
}

public void refBuilder() {

	Map<String, String> temp = new TreeMap<>();
	temp.put("1409S413", "2002H712");
	temp.put("0905A365", "0905A811");
	temp.put("0905A112", "2002H341");
	temp.put("2002H136", "2910h074");
	temp.put("1409S381", "0905A363");
	temp.put("2002H063", "0905A044");
	temp.put("0905A343", "2910h244");
	temp.put("1409S974", "1409S811");
	temp.put("2910h785", "2002H924");
	setCrossRef(temp);
}
public String itemToRef(String itemId) {
	String result = "";
	refBuilder();
	Map<String, String> items = getCrossRef();
	if(items.containsKey(itemId)) {
		result = items.get(itemId);
	}
	return result;
}


/**
 * Default constructor.
 */
public Reference() {
	// TODO Auto-generated constructor stub
}

/**
 * @see Filter#destroy()
 */
public void destroy() {
	// TODO Auto-generated method stub
}

/**
 * @see Filter#doFilter(ServletRequest, ServletResponse, FilterChain)
 */
```

```
	public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
			throws IOException, ServletException {
		HttpServletRequest hreq = (HttpServletRequest) request;
		Engine engine = Engine.getInstance();
		Map<String, Integer> cart = (Map<String, Integer>)
hreq.getSession().getAttribute("cart");


		boolean inCart = false;
		//items in to ref
		for(String id : cart.keySet()) {
			if(itemToRef(id) != "") {
				inCart = true;
			}
		}


		if (hreq.getRequestURI().matches(".*Cart.do")) {
			if (inCart == true)
			{
				List<ItemBean> iBeans = new ArrayList<>();
				List<ItemBean> sBeans = new ArrayList<>();

				for(String items : cart.keySet()) {
					if(getCrossRef().containsKey(items)) {
						try {
							iBeans.add(engine.getItem(items));
						} catch (Exception e) {
							e.printStackTrace();
						}
					}
				}

				for(ItemBean items : iBeans) {
					try {

sBeans.add(engine.getItem(itemToRef(items.getNumber())));
					} catch (Exception e) {
						e.printStackTrace();
					}
				}

				Map<ItemBean, Integer> viewableCart = null;
				try {
					viewableCart = engine.makeViewableCart(cart);
```

```
                                hreq.setAttribute("viewableCart", viewableCart);
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                        if (hreq.getParameter("updateCartButton") != null) {
                                String[] itemIds = hreq.getParameterValues("itemId");
                                String[] itemQuantities =
hreq.getParameterValues("quantityInput");
                                String[] deleteCheckboxes = null;
                                System.out.println(itemQuantities.length + " " +
itemIds.length);
                                if (hreq.getParameterValues("deleteCheckbox") !=
null) {
                                        deleteCheckboxes =
hreq.getParameterValues("deleteCheckbox");
                                }
                                Map<String, Integer> newCart = null;
                                try {
                                        newCart = engine.updateCart(cart, itemIds,
itemQuantities, deleteCheckboxes);
                                        hreq.getSession().setAttribute("cart",
newCart);
                                        try {
                                                viewableCart =
engine.makeViewableCart(newCart);
                                                hreq.setAttribute("viewableCart",
viewableCart);
                                        } catch (Exception e) {
                                                hreq.setAttribute("error", "Please
enter a valid input");
                                        }
                                } catch (Exception e) {
                                        hreq.setAttribute("error", "Please enter a
valid input");
                                }
                        }

                        if (engine.isCartEmpty(cart)) {
                                hreq.setAttribute("itemsCost", 0.0);
                                hreq.setAttribute("hstAmount", 0.0);
                                hreq.setAttribute("shippingCost", 0.0);
                        } else {
                                double itemsCost = engine.getItemsCost(viewableCart);

                                double shippingCost =
engine.getShippingCost(itemsCost);
```

```
                                double hstAmount = engine.getHstAmount(itemsCost,
shippingCost);
                                hreq.setAttribute("itemsCost", itemsCost);
                                hreq.setAttribute("hstAmount", hstAmount);
                                hreq.setAttribute("shippingCost", shippingCost);
                                hreq.setAttribute("suggestedList", iBeans);
                                hreq.setAttribute("suggestedCart", sBeans);
                        }

                        hreq.setAttribute("cart", hreq.getAttribute("cart"));
                        hreq.getServletContext().getRequestDispatcher("/
SuggestedCart.jspx").forward(request, response);
                  }

                  else
                  {
                        chain.doFilter(request, response);
                  }

            } else if (hreq.getRequestURI().matches("/eFoods/Search.do")) {
                  if (request.getParameter("cartButton") != null) { // If the add to
cart item is clicked
                        // We similarly resort the page


                        String item = request.getParameter("hiddenItemNo");
                        String quantity = request.getParameter("addQuantity");
                        try {
                              Map<String, Integer> newCart =
engine.addItemToCart(cart, item, quantity);
                              hreq.getSession().setAttribute("cart", newCart);
                        } catch (Exception e) {
                              System.out.println(e.getMessage());
                        }
                        hreq.getServletContext().getRequestDispatcher("/
Cart.do").forward(request, response);
                  }else {
                        chain.doFilter(request, response);
                  }
            }

      }

      /**
       * @see Filter#init(FilterConfig)
       */
```

```
	public void init(FilterConfig fConfig) throws ServletException {
		// TODO Auto-generated method stub
	}

}
```

Listeners

Analytics.java

```
package listeners;

import java.util.List;

import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;
import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpSessionAttributeListener;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;

/**
 * Watches events in the application to calculate analytics for the admin.
 *
 */
@WebListener
public class Analytics implements HttpSessionListener, HttpSessionAttributeListener,
ServletRequestListener {

	// When the session is created the time of the session is recorded in the
	// session.
	public void sessionCreated(HttpSessionEvent se) {
		HttpSession session = se.getSession();
		Long currentTime = System.currentTimeMillis();

		session.setAttribute("sessionStart", currentTime);
		session.setAttribute("userAddedToCart", false);
		session.setAttribute("userCheckedOut", false);
	}

	// When the user first adds an item to a cart, or checks out an order, the time
	// inbetween the session start and the event is recorded in the respective
lists
	// which are stored in the context.
	@SuppressWarnings("unchecked")
	public void requestInitialized(ServletRequestEvent sre) {
```

```
		HttpServletRequest httpReq = (HttpServletRequest)
sre.getServletRequest();
		String pageVisted = httpReq.getRequestURL().toString();
		HttpSession session = httpReq.getSession();

		if (pageVisted.matches(".*Catalog.do")) {

			boolean userAddedToCart = (boolean)
session.getAttribute("userAddedToCart");
			if (httpReq.getParameter("cartButton") != null && userAddedToCart
== false) {
				long timeTaken = getTimeTaken(session);

				List<Integer> timeBetweenAdd = (List<Integer>)
httpReq.getServletContext()
						.getAttribute("timeBetweenAdd");
				timeBetweenAdd.add((int) timeTaken);
				session.setAttribute("userAddedToCart", true);
			}

		} else if (pageVisted.matches(".*Order.do")) {

			boolean userCheckedOut = (boolean)
session.getAttribute("userCheckedOut");
			if (httpReq.getParameter("confirmOrderButton") != null &&
userCheckedOut == false) {
				long timeTaken = getTimeTaken(session);

				List<Integer> timeBetweenCheckout = (List<Integer>)
httpReq.getServletContext()
						.getAttribute("timeBetweenCheckout");
				timeBetweenCheckout.add((int) timeTaken);
				session.setAttribute("userCheckedOut", true);
			}
		}
	}

	// Calculates the seconds between the session start and the event.
	private long getTimeTaken(HttpSession session) {
		long currentTime = System.currentTimeMillis();
		long sessionStart = (long) session.getAttribute("sessionStart");

		long timeTaken = (currentTime - sessionStart) / 1000;
		return timeTaken;
	}
```

```
}
```

Init.java

```
package listeners;

import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpSessionAttributeListener;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;

import model.Engine;

/**
 * Initializes attributes upon context and session initialization.
 *
 */
@WebListener
public class Init implements ServletContextListener, HttpSessionListener,
HttpSessionAttributeListener {

	// Initializes the engine upon server creation, and adds the lists for
	// analytics.
	public void contextInitialized(ServletContextEvent sce) {
		Engine.getInstance();
		ServletContext context = sce.getServletContext();

		List<Integer> timeBetweenAdd = new LinkedList<>();
		List<Integer> timeBetweenCheckout = new LinkedList<>();

		context.setAttribute("timeBetweenAdd", timeBetweenAdd);
		context.setAttribute("timeBetweenCheckout", timeBetweenCheckout);
	}

	// Creates the cart in the session and sets authenticated to false.
	public void sessionCreated(HttpSessionEvent se) {
		Map<String, Integer> cart = new HashMap<>();
```

```
            se.getSession().setAttribute("cart", cart);
            se.getSession().setAttribute("authenticated", false);
      }

}
```

Middleware

Middleware.java

```
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;

import model.*;

public class Middleware {

      private static final String IN_PO_PATH = "/inPO/";
      private static final String OUT_PO_PATH = "/outPO/";

      private static final Class<OrderBean> ORDER_BEAN = OrderBean.class;
      private static final Class<ReportBean> REPORT_BEAN = ReportBean.class;

      private String poPath;
      private File inDir;
      private File outDir;

      private File[] poFiles;
      private File[] inFiles;

      private Marshaller reportMarshaller;
      private Unmarshaller orderUnMarshaller;

      public Middleware(File poDir) {
            this.poFiles = poDir.listFiles();
```

```
		setJAXB(ORDER_BEAN, REPORT_BEAN);
		setInOutDirFiles(poDir);
	}

	private void setJAXB(Class<OrderBean> orderBean, Class<ReportBean> reportBean)
{
		try {
			JAXBContext reportContext = JAXBContext.newInstance(reportBean);
			JAXBContext orderContext = JAXBContext.newInstance(orderBean);

			this.reportMarshaller = reportContext.createMarshaller();
			this.orderUnMarshaller = orderContext.createUnmarshaller();


this.reportMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
		} catch (JAXBException e) {
			System.err.println("Fatal JAXB error " + e.getMessage());
			System.exit(1);
		}
	}

	private void setInOutDirFiles(File poDir) {
		this.poPath = poDir.getPath();

		this.inDir = new File(poPath + IN_PO_PATH);
		this.outDir = new File(poPath + OUT_PO_PATH);

		if (!inDir.isDirectory() || !outDir.isDirectory()) {
			throw new IllegalArgumentException("There is no inPO or outPO
directory in the PO folder, terminating.");
		}

		this.inFiles = inDir.listFiles();
	}

	// TODO The files should be moved right after being read into the array or
after
	// report made

	/**
	 * O(n) time
	 *
	 * @return
	 * @throws Exception
	 */
	public List<OrderBean> getInboxOrders() throws Exception {
```

```
		List<OrderBean> orderList = new ArrayList<>();

		if (inFiles.length == 0) {
			throw new Exception("Inbox folder is empty, terminating.");
		}

		for (File fileName : inFiles) {
			OrderBean order = (OrderBean)
orderUnMarshaller.unmarshal(fileName);
			orderList.add(order);
		}

		return orderList;
	}

	public Map<String, TotalItemsBean> consolidateOrders(List<OrderBean> orderList)
{
		Map<String, TotalItemsBean> quantityMap = new HashMap<>();

		for (OrderBean order : orderList) {
			List<ItemBean> orderItems = order.getItems();

			for (ItemBean orderItem : orderItems) {
				String orderItemNumber = orderItem.getNumber();
				String orderItemName = orderItem.getName();
				int orderItemQty = orderItem.getQuantity();

				if (!quantityMap.containsKey(orderItemNumber)) {
					TotalItemsBean totalItem = new TotalItemsBean();

					totalItem.setNumber(orderItemNumber);
					totalItem.setName(orderItemName);
					totalItem.setQuantity(orderItemQty);

					quantityMap.put(orderItemNumber, totalItem);
				} else {
					TotalItemsBean totalItem =
quantityMap.get(orderItemNumber);
					totalItem.setQuantity(totalItem.getQuantity() +
orderItemQty);
					quantityMap.put(orderItemNumber, totalItem);
				}
			}
		}

		return quantityMap;
```

```
	}

	public ReportBean makeReport(Map<String, TotalItemsBean> quantityMap) {
		ReportBean report = new ReportBean();

		List<TotalItemsBean> totalItemsList = new
LinkedList<>(quantityMap.values());
		report.setItems(totalItemsList);

		return report;
	}

	public void marshallReport(ReportBean report) throws JAXBException, IOException
{
		File reportFile = new File(poPath + "/report" + (poFiles.length - 1) +
".xml");
		reportFile.createNewFile();

		reportMarshaller.marshal(report, reportFile);

		moveCompletedOrders();

	}

	private void moveCompletedOrders() {

		for (File inFile : inFiles) {
			File outFile = new File(outDir.getPath() + "/" +
inFile.getName());
			inFile.renameTo(outFile);
		}

	}

	public File getInDir() {
		return inDir;
	}

	public File getOutDir() {
		return outDir;
	}

	public static void main(String[] args) {

		if (args.length != 1) {
			System.err.println("Usage: java Middleware <PO Folder Path>");
```

```
                        System.exit(1);
                }

                File poDir = new File(args[0]);

                if (!poDir.isDirectory()) {
                        System.err.println("There is no PO directory, exiting");
                        System.exit(1);
                }

                try {
                        Middleware b2c = new Middleware(poDir);
                        List<OrderBean> orderList = b2c.getInboxOrders();
                        Map<String, TotalItemsBean> quantityMap =
b2c.consolidateOrders(orderList);
                        ReportBean report = b2c.makeReport(quantityMap);
                        b2c.marshallReport(report);
                } catch (Exception e) {
                        System.err.println(e.getMessage());
                        System.exit(1);
                }

                System.out.println("Program complete, exiting.");
        }

}
```

MiddlewareTest.java

```
import static org.junit.jupiter.api.Assertions.*;

import java.io.File;
import java.util.List;

import javax.xml.bind.JAXBException;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;

import model.OrderBean;
```

```
class MiddlewareTest {

	static final String PO_FOLDER = System.getProperty("user.home") + "/PO/";
	static final File PO_FILE = new File(PO_FOLDER);
	static Middleware b2c;

	@BeforeAll
	static void setUpBeforeClass() throws Exception {
		b2c = new Middleware(PO_FILE);
	}

	@AfterAll
	static void tearDownAfterClass() throws Exception {
	}

	@BeforeEach
	void setUp() throws Exception {
	}

	@AfterEach
	void tearDown() throws Exception {
	}

	@Test
	void testMiddleware() {

		try {
			Middleware testConstruct = new Middleware(PO_FILE);
			assertTrue(true);
		} catch (Exception e) {
			fail("Error thrown " + e.getMessage());
		}
	}

	@ParameterizedTest
	@ValueSource(strings = { "bad directory", "/cs/adamzis/test", "/eecs/home/
adamzis/PO/inPO", "505",
			"/eecs/home/adamzis/PO/ouPO" })
	void testMiddlewareException(String nonsense) {
		try {
			File dummyfile = new File(nonsense);
			Middleware badMiddle = new Middleware(dummyfile);

			fail("No exception thrown");
		} catch (IllegalArgumentException e) {
			assertTrue(true);
```

```
		} catch (Exception e) {
			fail("Wrong exception thrown");
		}

	}

	@Test
	void testListPoFiles() throws JAXBException {
		List<OrderBean> returnedOrders = b2c.listInboxFiles();
		assertTrue(!returnedOrders.isEmpty());
	}

}
```

ReportBean.java

```
import java.util.List;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlElementWrapper;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "report")
@XmlAccessorType(XmlAccessType.FIELD)
public class ReportBean {

	@XmlElementWrapper
	@XmlElement(name = "item")
	private List<TotalItemsBean> items;
}
```

TotalItemsBean.java

```
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;

@XmlAccessorType(XmlAccessType.FIELD)
public class TotalItemsBean {

	@XmlAttribute
	int number;

	int name;
	int quantity;
```

```
}
```