

.....



Foods R Us

An E-Commerce Website

Designed and Developed by:

Haseeb Ahmad (cse: ahmadhas)

Talha Mahmood (cse: talha746)

Submitted as the group project for the course

EECS4413 - Building E-Commerce Systems

Prof. H. Roumani - Fall 2018

Table Of Contents

Design.....	4
Implementation.....	7
The Team.....	9
Output Samples.....	11
Acknowledgement.....	12
The Source Code.....	13
B2C Model.....	14
Model.....	15
Account.....	19
Catalog.....	20
Category.....	21
Item.....	23
Order.....	27
DAO.....	31
ConnectionFactory.....	32
DAO.....	33
CategoryDAO.....	37

ItemDAO.....	40
OrderDAO.....	43
Helpers.....	45
Utils.....	46
B2C Controller.....	48
Home.....	50
Cart.....	52
Checkout.....	53
PurchaseOrders.....	55
Admin.....	58
API.....	60
Product.....	61
CartApi.....	64
B2C Analytics.....	68
Monitor.....	69
ComputeAnalytics.....	76
B2C Adhoc.....	81
AuthFilter.....	82
B2C View.....	85
web.xml.....	86
Includes.....	87
Header.....	88
Nav.....	89
Footer.....	91
Error.....	93
Item.....	94
CatItemsCard.....	98

Pages.....	101
Home.....	102
Admin.....	106
Search.....	109
Cart.....	111
Checkout.....	115
PurchaseOrders.....	119
Error404.....	121
JavaScript.....	122
main.js.....	123
CSS.....	130
main.css.....	131
Middleware.....	132
Main.....	133
Model.....	134
Engine.....	135
Report.....	143
Product.....	144
Helpers.....	146
Utils.....	147

Design

B2C Design

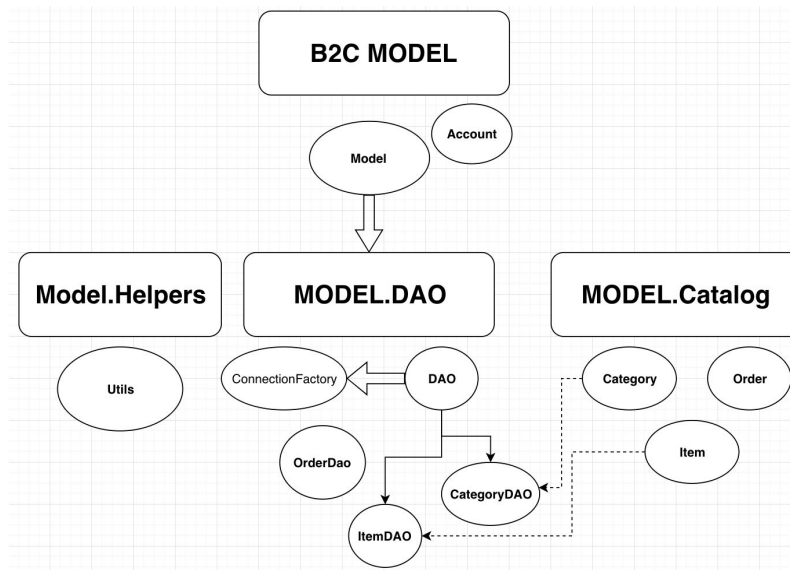
The main design pattern used in this project is Model View Controller (MVC).

The View package is divided into the “includes” directory (or partials) and pages. The “Includes” directory contains the view fragments used in the each page under “pages” directory which is served by the controller for the specific route for each page. We decided this because the partials, like the navigation bar for example is used on all the pages and we did not want to keep on copying all the code over on each page and same thing with items, that catalog is used in multiple tabs on the main page for each category. This reusability of code makes reading code easier and can help when expanding or changing the parts of the app.

In the Model, there is a connection to database, computations and global model class using singleton pattern in order to not create many instances when using it in controller.


The connection to database is in the dao package in the model which has a separate connection Factory (ConnectionFactory.java) being used in the DAO class which is then inherited by the ItemDAO and CategoryDAO. These classes populate the item and category beans. The DAO.java class is abstract and uses generics. The reason it is abstract is because we not only have to implement some methods that are accessible to the child classes that can be changed later, but we also had to make children classes implement the abstract methods used in the already created methods. The last dao class is the OrderDAO class which read and check if orders exist in the given directory and creates new order files in that directory. You can search for orders in the given directory using the user’s name which is part of the Purchase Order file name. There is also a helpers package in the model which holds Utils class containing static methods and constructor to be reused and standalone methods in the model as well as controller.

Below is the diagram of the model.



Finally, in the Controller package we have all our servlets as well as our api. We have an admin servlet setup in case we need to use an admin page in the future (to show all the analytics), and then we have our Cart servlet for our cart page, the Checkout servlet for our checkout page, Home servlet for the Home page (our main landing page), Item servlet in case we need to view an item individually (we didn't implement this fully as there was no need but we created it in case we want to in the future), PurchaseOrders servlet for when viewing a purchase order page (after checkout we provide a link to view purchase order as well as view all purchase orders made by the logged in user), and finally the Search servlet which is for the search results page. Then in the api folder, we have a CartApi servlet and a Product servlet. The CartApi Servlet does all of the cart actions which are adding item to cart as well as updating items' quantity and deleting items. The Product Servlet isn't being used in the app, but for future purposes if we want to change restful api, then we can get and show the item using api requests.

The Monitor class in analytics package is used, when a new session is created, to not only do analytics, but to store the order and user info in the session in order to be used throughout the app in the controller, especially to store items in the cart and for checkout the cart, so we know if any items are in the cart from any page within the app.



Design for the open authentication: So the authentication server's job is to redirect traffic to the open auth server in order to validate the clients identity. The purpose of this is so that the user/client can feel more comfortable about the security of their login information, as it is directly validated from the EECS server in our case (GMail open auth is one of the most commonly used open authentication servers as another example).

The way we implemented this open auth server is using the professors open auth server from Project B in a servlet filter called AuthFilter.java. This servlet is called whenever the user/client tries to checkout but has not yet logged in or simply logs in from a fresh visit. An EECS login is then prompted and we can get both the name of the user and their username, and once logged in, the login button changes to show "Hello (User's name)" instead (where User's name is store in the string variable "String userName").

Middleware Design

So our middleware is split into three packages called Middleware (with the Main.java file that calls the engine), helpers package (with a utils.java class that has two methods, one to transform xsl to html and the other to simply round numbers), and finally a model package (that contains the Engine.java, Product.java, and Report.java classes). The Product class is to simply define a product as an xml "item" element, and the Report class is to define a procurement report as an xml "report" element.

The way that the middleware connects to B2C is through the outputted purchase orders (saved as files) created by the B2C and then processed by the middleware after it has an idea of what it is reading (the products are specified in Product.java for this reason).

In the Engine, each purchase order file (xml) is processed one by one (by checking the directory specified) and then creates a procurement report (which is needed for the B2B) as an xml or json file that has each item ordered only appearing once with its quantity equal to the total quantity from all the purchase orders added up.

Implementation


We ran into a lot of implementation issues along the way and in this section we will go over some of the decisions we made in order to get around those issues and fix them.

Here are all the issues we ran into:

1. When successfully logging in, we are redirected to an page that only contains null, but this only happens sometimes.
2. Items are duplicated when adding to shopping cart instead of updating that item's quantity.
3. The search option was not working.
4. No sign out option for the user.
5. The update button at the bottom of the cart does not update quantity for all the items but the update button on the right of each item does update the quantity for that specific item.
6. Trouble implementing confirmation page after checking out in the cart and before finalizing the purchase order.

We figured all these issues out when testing our webapp, and we were constantly testing after implementing each part in increments so we had an agile approach. Testing was a major factor for us to make sure everything was working as it was supposed to and followed the specifications. We simply ran our webapp on localhost server and loaded it up on firefox in our virtualbox and went through all the pages and simulated a users experience to figure out how it would go and analyze and inspect things along the way. We would simulate different types of users such as a user that would only try to log in and then view items and exit, then another user who would add multiple items to cart, attempt to checkout and then get authenticated and finalize his/her order, another user that would just search for items but never buy, etc. We tested multiple scenarios so that we would cover all bases and see if each feature was working as intended.

Now we will explain what fixes we had for the issues above and how we fixed them:

- 
1. This issue only happened on our laptops and only on first sign in, we tested multiple times again on the lab workstations and could not reproduce this issue.
 2. So we changed our design so that we check if the item is already in cart when trying to add it and if it is, it simply increases its quantity by 1 instead.
 3. Fixed search option to only sort through items and so that the search key does not have to be exactly the same as the item name or number, it could be similar to (we used "LIKE" when searching the database).
 4. Ended up not needed a sign out feature, if you need to login as a different user, you can simply start a new session.
 5. Fixed so that it simply makes multiple api requests to cart for the quantity of each item
 6. At the cart page, when about to check out, it leads to a confirmation page before the order is finalized in order to give the user one final chance to look at their cart items and make sure everything is in order. We created a new request variable that checks if the user clicked confirm or not to enforce this (if it is not checked then it does not finalize the order).

In the end we were able to fix all of these implementation issues and now there is no shortcomings in our webapp in regards to the specifications.

The Team

The work was mostly done together when we held our meetings. We planned and designed the system together in person in these meetings as well as assisted each other when one of us was writing the code, so we all had input in it. We tried to divide the responsibilities of the overall project evenly and have detailed how below.

The team met up regularly, at least once a week since the project was assigned but usually twice a week (sometimes off campus). The meetings were where we planned what we needed to do, helped each other implement it, test the system as we went along, and plan what else was needed (for the next time).

Lessons learned: We learned that it is more effective to work together in person rather than working separately as we are productive when we meet up in person and get things done pretty quickly. Having meetups the way we did really helped accelerate our progress on this project and so we learned how to work better as a team because of it.

We also improved our collaboratory skills in regards to creating a project with multiple components in a group which should be beneficial for us in the future. We also uploaded and shared our code with each other on github so that was also a good learning experience as github is widely used in the software industry when working on large projects.

We divided the work responsibility as follows:

Haseeb: implemented the front end and setup the whole system (Creating the project etc), also led B2C implementation when we worked on it as a group. Setup the AUTH server as AuthFilter.java in adhoc package (using Professors server). Fixed a lot of our implementation issues.

Talha: Implemented the bean files, the dao files, assisted with the overall B2C and middleware implementation when working on it as a group, led the testing and helped fix implementation issues, led writing on the report. Helped with the front end and middleware implementation as well.

We both knew what the other was doing and were learning it as the other was implementing it, we would also each study the overall system as it was being developed and would explain what we did if done on our own time. Since we did it in this collaboratory way, we both understand all aspects of each other's work and of the overall system.

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

Haseeb Ahmad:



Talha Mahmood:



Output Samples

Here is one P/O example, it is named: poahmadhas_11.xml and here is how the xml file looks:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><?xml-stylesheet type="text/xsl" href="res/xml/P0.xsl"?>
<order id="13" submitted="2018-11-18">
  <customer account="ahmadhas">
    <name>Haseeb Ahmad</name>
  </customer>
  <grandTotal>23.37</grandTotal>
  <HST>2.69</HST>
  <items>
    <item number="1409S039">
      <catID>5</catID>
      <name>M8 Peanut Ice Cream</name>
      <price>4.5</price>
      <quantity>1</quantity>
      <extended>4.5</extended>
    </item>
    <item number="1409S123">
      <catID>5</catID>
      <name>E8 Nuts Ice Cream</name>
      <price>6.12</price>
      <quantity>1</quantity>
      <extended>6.12</extended>
    </item>
    <item number="1409S250">
      <catID>5</catID>
      <name>G8 Praline Ice Cream</name>
      <price>5.06</price>
      <quantity>1</quantity>
      <extended>5.06</extended>
    </item>
  </items>
  <shipping>5.0</shipping>
  <total>15.68</total>
</order>
```

Here is how it appears on the webapp after an order:

[Back to Shop](#)

Hello Haseeb Ahmad

Your order has been recieved.

Account: ahmadhas

Submitted on: 2018-11-18

Product Number	Product Name	Price	Extended Price	Quantity
1409S039	M8 Peanut Ice Cream	4.5	4.5	1
1409S123	E8 Nuts Ice Cream	6.12	6.12	1
1409S250	G8 Praline Ice Cream	5.06	5.06	1

Summary

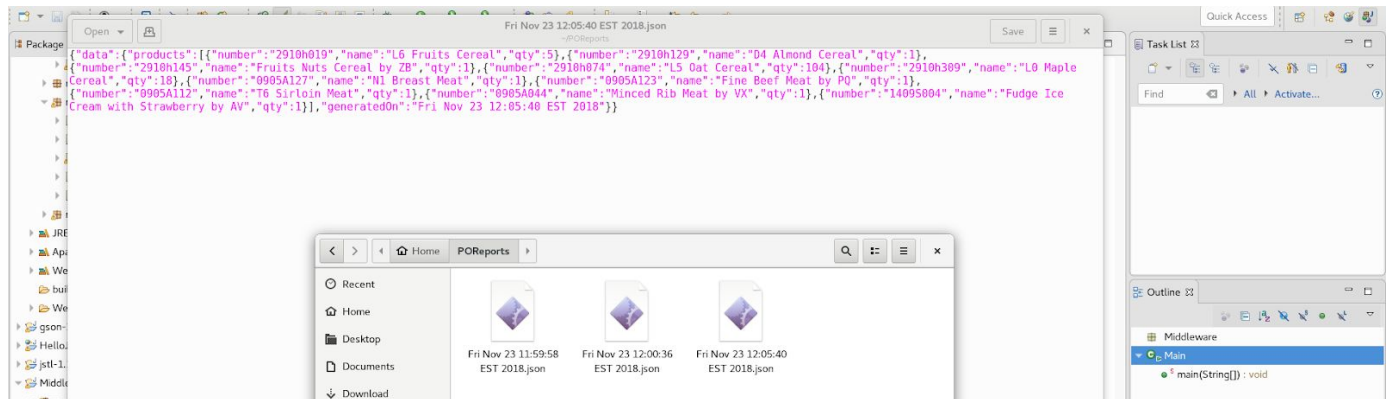
Total: 15.68

Shipping: 5.0

HST: 2.69

Grand Total: 23.37

Here is one procurement report example, it is named: Fri Nov 23 12_05_40 EST 2018.json and here are its contents and where the purchase order files are stored on disk:



So they are saved in the a folder called POReports from the home directory (we set this ourselves). The format of the file name is that it starts the date then the time and then ends in (.json) or (.xml). These procurement reports are made for the purpose of the B2B app processing it and doing what it needs to with the given information (fulfilling all those orders internally) but we do not need to take care of B2B implementation.

Acknowledgment

For the front-end styles of our app, we used Bulma CSS (bulma.io) library, as well as the images used are taken from flaticon.com. We have included a "Made By Bulma" image on our app in the footer.

Source Code

Below are the source files of the project.



Model

Model.java

```
1 package model;
2
3 import java.io.FileWriter;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 public class Model {
22     private static Model instance = null;
23     private static ItemDAO itemDao;
24     private static CategoryDAO catDao;
25
26     private Model() {
27         try {
28             itemDao = new ItemDAO();
29             catDao = new CategoryDAO();
30         } catch (Exception e) {
31             System.out.println("DB ERROR: " + e.getMessage());
32             e.printStackTrace();
33         }
34     }
35
36     public static Model getInstance() {
37         if (instance == null) {
38             instance = new Model();
39         }
40         return instance;
41     }
42
43     public List<Item> getFoods() throws Exception {
44         return getFoods("0");
45     }
46
47     public List<Item> getFoods(String limit) throws Exception {
48         int l = Integer.parseInt(limit);
49         return itemDao.getAll(l);
50     }
51
52     public Item getFood(String val) throws Exception {
53         return this.getFoodBy("id", val);
54     }
55
56     public Item getFoodBy(String by, String val) throws Exception
57     {
58         return this.getFoodBy(by, val, false);
59     }
```


Model.java

```
60     public Item getFoodBy(String by, String val,
61         Boolean like) throws Exception {
62         return itemDao.findOneBy(by, val, like);
63     }
64
65     public List<Item> getFoodsByMultiple(String val) throws
Exception {
66         return itemDao.getAllByMultiple(val, true, null);
67     }
68
69     public List<Item> getFoodsBy(String by,
70         String val) throws Exception {
71         return itemDao.getAllBy(by, val, true);
72     }
73
74     public List<Item> getFoodsBy(String by,
75         String val, Boolean like) throws Exception {
76         return getFoodsBy(by, val, like, "0");
77     }
78
79     public List<Item> getFoodsBy(String by, String val,
80         Boolean like, String limit) throws Exception {
81         int l = Integer.parseInt(limit);
82         return itemDao.getAllBy(by, val, like, l);
83     }
84
85     public List<Category> getCategories() throws Exception {
86         return catDao.getAll();
87     }
88
89     public Category getCategory(String cat) throws Exception {
90         return catDao.findOne(cat);
91     }
92
93     public Map<String, List<Item>> getCatNameWithFoods() throws
Exception {
94         Map<String, List<Item>> result = new HashMap<String,
List<Item>>();
95
96         for (Category cat: this.getCategories()) {
97             List<Item> items = new ArrayList<Item>();
98             for (Item item: this.getFoods()) {
99                 if (cat.getId() == item.getCatID()) {
100                     items.add(item);
```

Model.java

```
101         result.put(cat.getName(), items);
102     }
103 }
104 }
105
106     return result;
107 }
108
109     public Map<Category, List<Item>> getCatsWithFoods() throws
Exception {
110         Map<Category, List<Item>> result = new HashMap<Category,
List<Item>>();
111
112         for (Category cat: this.getCategories()) {
113             List<Item> items = new ArrayList<Item>();
114             for (Item item: this.getFoods()) {
115                 if (cat.getId() == item.getCatID()) {
116                     items.add(item);
117                     result.put(cat, items);
118                 }
119             }
120         }
121
122         return result;
123     }
124
125     public synchronized StringWriter createP0(String filePath,
126         int orderNum, String xslFilename,
127         Order order, Account user) throws Exception
128     {
129         OrderDAO orderDao = new OrderDAO(filePath);
130         String fileName =
orderDao.getOrderFileName(user.getUsername(), orderNum);
131
132         JAXBContext jaxbContext =
JAXBContext.newInstance(order.getClass());
133         Marshaller marshaller = jaxbContext.createMarshaller();
134         marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
Boolean.TRUE);
135         marshaller.setProperty(Marshaller.JAXB_FRAGMENT,
Boolean.TRUE);
136
137         StringWriter sw = new StringWriter();
138         sw.write("<?xml version=\"1.0\" encoding=\"UTF-8\"")
```

Model.java

```
standalone=\"yes\"?>");
139     sw.write("<?xml-stylesheet type=\"text/xsl\" href=\"\" +
xslFilename + "\"?>\n");
140
141     marshaller.marshal(order, new StreamResult(sw));
142     FileWriter orderFileWrite =
orderDao.getFileWriter(fileName);
143     orderFileWrite.write(sw.toString());
144     orderFileWrite.close();
145
146     return sw;
147 }
148
149
150 public static void main(String[] args) {
151     Model model = Model.getInstance();
152     try {
153         System.out.println(model.getCategories().toString());
154         System.out.println(model.getFoods().toString());
155     } catch (Exception e) {
156         System.out.println("ERROR: " + e.getMessage());
157     }
158 }
159 }
160
```

Account.java

```
1 package model;
2
3 import javax.xml.bind.annotation.XmlAttribute;
4
5
6
7 @XmlElement(name="customer")
8 public class Account {
9     private String username;
10    private String name;
11
12    public Account() {
13
14    }
15
16    public Account(String name, String username) {
17        this.setName(name);
18        this.setUsername(username);
19    }
20
21    @XmlElement(name="name")
22    public String getName() {
23        return name;
24    }
25
26    public void setName(String name) {
27        this.name = name;
28    }
29
30    @XmlAttribute(name="account")
31    public String getUsername() {
32        return username;
33    }
34
35    public void setUsername(String username) {
36        this.username = username;
37    }
38 }
39
```



Model.Catalog

Category.java

```
1 package model.catalog;
2
3 import java.sql.Blob;
4
5 public class Category {
6
7     private String name, description;
8     private int id;
9     private Blob picture;
10
11     public Category() {
12     }
13
14     public Category(String name, String description, int id, Blob
picture) {
15         setName(name);
16         setDescription(description);
17         setId(id);
18         setPicture(picture);
19     }
20
21     public String getName() {
22         return name;
23     }
24
25     public void setName(String name) {
26         this.name = name;
27     }
28
29     public String getDescription() {
30         return description;
31     }
32
33     public void setDescription(String description) {
34         this.description = description;
35     }
36
37     public int getId() {
38         return id;
39     }
40
41     public void setId(int id) {
42         this.id = id;
43     }
44 }
```

Category.java

```
44
45     public Blob getPicture() {
46         return picture;
47     }
48
49     public void setPicture(Blob picture) {
50         this.picture = picture;
51     }
52
53
54     public String toString() {
55         return String.format("Name: %s | Description: %s | ID: %s |
Picture: %s \n", name, description, id, picture);
56         //fix this
57     }
58
59
60
61 }
62
```

Item.java

```
1 package model.catalog;
2
3 import javax.xml.bind.annotation.XmlAttribute;
4
5 @XmlRootElement(name="item")
6 public class Item {
7
8     private String name, number, unit;
9
10    private double price, totalPrice;
11    private int quantity, catID;
12
13    public Item() {
14    }
15
16    public Item(String name, double price, int quantity, String
17    number, String unit) throws Exception {
18        this(name, price, quantity, number, 0, unit);
19    }
20
21    public Item(String name, double price, int quantity, String
22    number, int catId, String unit) throws Exception {
23        setName(name);
24        setPrice(price);
25        setQuantity(quantity);
26        setNumber(number);
27        setCatID(catID);
28        setUnit(unit);
29    }
30
31    @XmlElement(name="name")
32    public String getName() {
33        return name;
34    }
35
36    public void setName(String name) {
37        this.name = name;
38    }
39
40    @XmlElement(name="price")
41    public double getPrice() {
42        return Utils.round(price, 2);
43    }
44
45 }
46
```


Item.java

```
47     public void setPrice(double price) throws Exception {
48         if (price < 0.0) {
49             throw new IllegalArgumentException("Price < 0.0");
50         }
51         this.price = Utils.round(price, 2);
52     }
53
54     @XmlElement(name="quantity")
55     public int getQuantity() {
56         return quantity;
57     }
58
59     public void setQuantity(int quantity) throws Exception {
60         if (quantity < 0) {
61             throw new IllegalArgumentException("Qty < 0");
62         }
63         this.quantity = quantity;
64     }
65
66     public void setQuantity(String quantity) throws Exception {
67         int qty = Integer.parseInt(quantity);
68         this.setQuantity(qty);
69     }
70
71     @XmlAttribute(name="number")
72     public String getNumber() {
73         return number;
74     }
75
76     public void setNumber(String number) {
77         this.number = number;
78     }
79
80     public int getCatID() {
81         return catID;
82     }
83
84     public void setCatID(int catID) {
85         this.catID = catID;
86     }
87
88     public String getUnit() {
89         return unit;
90     }
```

Item.java

```
91
92     public void setUnit(String unit) {
93         this.unit = unit;
94     }
95
96     public void setTotalPrice(double totalPrice) {
97         this.totalPrice = Utils.round(totalPrice, 2);
98     }
99
100     @XmlElement(name="extended")
101     public double getTotalPrice() {
102         return Utils.round(totalPrice, 2);
103     }
104
105     public void increaseQty() {
106         this.quantity++;
107     }
108
109     public void increaseQty(int qty) {
110         this.quantity += qty;
111     }
112
113     public void increaseQty(String qtyStr) throws Exception {
114         int qty = Integer.parseInt(qtyStr);
115         this.setQuantity(qty);
116     }
117
118     public double computeTotalPrice() {
119         this.setTotalPrice(this.price * this.quantity);
120         return Utils.round(this.totalPrice, 2);
121     }
122
123     public String toString() {
124         return String.format("Name: %s | Price: %s | Quantity: %s
| Number: %s | CatId: %s | Unit: %s \n", name, price, quantity,
number, catID, unit);
125     }
126
127     @Override
128     public boolean equals(Object obj) {
129         if (this == obj) return true;
130         if (obj == null) return false;
131         if (!(obj instanceof Item)) return false;
132
```

Item.java

```
133     Item other = (Item) obj;
134     if (!this.number.equals(other.number)) {
135         return false;
136     }
137
138     return true;
139 }
140
141
142 }
143
```

Order.java

```
1 package model.catalog;
2
3 import java.text.SimpleDateFormat;
14
15 @XmlRootElement(name="order")
16 public class Order {
17     private static final double HST_COST = 0.13;
18     private static final double SHIPPING_COST = 5.0;
19     private static final double SHIPPING_BUFFER_COST = 100.0;
20     private static final SimpleDateFormat sdf = new
SimpleDateFormat("yyyy-MM-dd");
21
22     private List<Item> items;
23     private Account customer;
24     private double total;
25     private double shipping;
26     private double HST;
27     private double grandTotal;
28
29     private int id;
30     private String submitted;
31
32     public Order() {
33         this(new ArrayList<Item>());
34     }
35
36     public Order(List<Item> items) {
37         this(0, items);
38     }
39
40     public Order(int id, List<Item> items) {
41         this.id = id;
42         this.items = items;
43     }
44
45     @XmlElementWrapper(name="items")
46     @XmlElement(name="item")
47     public List<Item> getItems() {
48         return items;
49     }
50
51     public void setItems(List<Item> items) {
52         this.items = items;
53     }
```

Order.java

```
54
55 @XmlElement(name="customer")
56 public Account getCustomer() {
57     return customer;
58 }
59
60 public void setCustomer(Account customer) {
61     this.customer = customer;
62 }
63
64 @XmlElement(name="total")
65 public double getTotal() {
66     return Utils.round(total, 2);
67 }
68
69 public void setTotal(double total) {
70     this.total = Utils.round(total, 2);
71 }
72
73 @XmlElement(name="shipping")
74 public double getShipping() {
75     return Utils.round(shipping, 2);
76 }
77
78 public void setShipping(double shipping) {
79     this.shipping = Utils.round(shipping, 2);
80 }
81
82 @XmlElement(name="HST")
83 public double getHST() {
84     return Utils.round(HST, 2);
85 }
86
87 public void setHST(double hST) {
88     HST = Utils.round(hST, 2);
89 }
90
91 @XmlElement(name="grandTotal")
92 public double getGrandTotal() {
93     return Utils.round(grandTotal, 2);
94 }
95
96 public void setGrandTotal(double grandTotal) {
97     this.grandTotal = Utils.round(grandTotal, 2);
```

Order.java

```
98     }
99
100    @XmlAttribute(name="id")
101    public int getId() {
102        return id;
103    }
104
105    public void setId(int id) {
106        this.id = id;
107    }
108
109    @XmlAttribute(name="submitted")
110    public String getSubmitted() {
111        return submitted;
112    }
113
114    public void setSubmitted(String submitted) {
115        this.submitted = submitted;
116    }
117
118    public void setSubmitted(Date submitted) {
119        this.submitted = sdf.format(submitted);
120    }
121
122    public synchronized void computeGrandTotal() {
123        this.setGrandTotal(this.getTotal() + this.getHST() +
124        this.getShipping());
125    }
126
127    public synchronized void computeTotalHST() {
128        this.setHST((this.getTotal() + this.getShipping()) *
129        HST_COST);
130    }
131
132    public synchronized void computeShipping() {
133        if (this.getTotal() > 0 && this.getTotal() <=
134        SHIPPING_BUFFER_COST) {
135            this.setShipping(SHIPPING_COST);
136        } else {
137            this.setShipping(0.0);
138        }
139    }
140
141    public synchronized void computeTotalCost() {
```

Order.java

```
139         double total = 0.0;
140         for (Item it: this.getItems()) {
141             total += it.computeTotalPrice();
142         }
143         this.setTotal(total);
144     }
145
146     public synchronized void computeAllCosts() {
147         this.computeTotalCost();
148         this.computeTotalHST();
149         this.computeShipping();
150         this.computeGrandTotal();
151     }
152
153     public synchronized void clearCart() {
154         this.items.clear();
155     }
156
157     public synchronized void clearOrder() {
158         this.clearCart();
159         this.computeAllCosts();
160     }
161 }
162
```



Model.DA0

ConnectionFactory.java

```
1 package model.dao;
2
3 import java.sql.Connection;
4
5
6
7
8 public class ConnectionFactory {
9     public static final String DB_HOST = "localhost";
10    public static final String DB_PORT = "1527";
11    public static final String DB_NAME = "EECS";
12    public static final String DB_USER = "student";
13    public static final String DB_PASS = "secret";
14
15    public static String DB_SCHEMA = "roumani";
16
17    public static final String DB_URL = String.format(
18        "jdbc:derby://%s:%s/%s;user=%s;password=%s",
19        DB_HOST, DB_PORT, DB_NAME, DB_USER, DB_PASS);
20
21    public static final String DB_DRIVER =
22        "org.apache.derby.jdbc.ClientDriver";
23
24    public static Connection getConn() throws Exception {
25        Class.forName(DB_DRIVER).newInstance();
26        Connection con = DriverManager.getConnection(DB_URL);
27        if (DB_SCHEMA != null && DB_SCHEMA != "")
28            con.createStatement().executeUpdate("set schema " +
29                DB_SCHEMA);
30        return con;
31    }
32
33    public static void closeConn(ResultSet r, Statement s,
34        Connection c) throws Exception {
35        r.close();
36        s.close();
37        c.close();
38    }
39 }
```

DAO.java

```
1 package model.dao;
2
3 import java.sql.Connection;
4
5
6
7
8
9
10
11 public abstract class DAO<T> {
12     protected String TABLE_NAME;
13     protected static Connection conn;
14
15     public DAO() throws Exception {
16         this("");
17     }
18
19     public DAO(String tableName) throws Exception {
20         conn = ConnectionFactory.getConn();
21         setTableName(tableName);
22     }
23
24     public void setTableName(String name) {
25         TABLE_NAME = name;
26     }
27
28     public String getTableName() {
29         return TABLE_NAME;
30     }
31
32     public String getAllQuery() {
33         return String.format("select * from %s", getTableName());
34     }
35
36     public List<T> getAll() throws Exception {
37         return getAll(0);
38     }
39
40     public List<T> getAll(Integer limit) throws Exception {
41         Connection conn = ConnectionFactory.getConn();
42         List<T> result = new ArrayList<T>();
43         String limitQuery = "";
44         // String limitQuery = (limit != null && limit > 0) ? "limit
45         " + limit : "";
46         String query = String.format("%s %s", getAllQuery(),
47         limitQuery);
48         PreparedStatement stmt = conn.prepareStatement(query);
49         ResultSet r = stmt.executeQuery();
```

DAO.java

```

49
50     while (r.next()) {
51         T i = createBean(r);
52         if (!result.contains(i)) {
53             result.add(i);
54         }
55     }
56
57     ConnectionFactory.closeConn(r, stmt, conn);
58     return result;
59 }
60
61     public List<T> getAllBy(String by, String val, Boolean like)
62     throws Exception {
63         return getAllBy(by, val, like, 0);
64     }
65
66     public List<T> getAllBy(String by, String val, Boolean like,
67     Integer limit) throws Exception {
68         Connection conn = ConnectionFactory.getConn();
69         List<T> result = new ArrayList<T>();
70         String limitQuery = "";
71         // String limitQuery = (limit != null && limit > 0) ? "limit
72         " + limit : "";
73         String equal = like ? "like" : "=";
74
75         val = like ? "'" + val + "'" : val;
76         by = this.convertToColumnName(by);
77
78         for (String[] str: getColumns().values()) {
79             if (str[0] == by) {
80                 if (str[1] == "str") {
81                     if (val.charAt(0) != '\\' &&
82                     val.charAt(val.length()-1) != '\\') {
83                         val = "'" + val + "'";
84                     }
85                 }
86             }
87         }
88
89         String query = String.format("%s where %s %s %s %s",
90         getAllQuery(), by, equal, val, limitQuery);
91
92         PreparedStatement stmt = conn.prepareStatement(query);

```

DAO.java

```

88         ResultSet r = stmt.executeQuery();
89
90         while (r.next())
91         {
92             T i = createBean(r);
93             if (!result.contains(i)) {
94                 result.add(i);
95             }
96         }
97         ConnectionFactory.closeConn(r, stmt, conn);
98         return result;
99     }
100
101     public List<T> getAllByMultiple(String val, Boolean like,
Integer limit) throws Exception {
102         Connection conn = ConnectionFactory.getConn();
103         List<T> result = new ArrayList<T>();
104         String limitQuery = "";
105 //         String limitQuery = (limit != null && limit > 0) ? "limit
" + limit : "";
106         String equal = like ? "like" : "=";
107
108         val = like ? "'" + val + "'" : val;
109
110         String whereQuery = "";
111
112         int idx=0;
113         for (String[] str: getSearchColumns()) {
114             String or = "OR";
115             if (str[1] == "str") {
116                 if (val.charAt(0) != '\\' &&
val.charAt(val.length()-1) != '\\') {
117                     val = "'" + val + "'";
118                 }
119             }
120
121             if (idx == getSearchColumns().size()-1) {
122                 or = "";
123             }
124
125             whereQuery += String.format(" %s %s %s %s ", str[0],
equal, val, or);
126             idx++;
127         }

```

DAO.java

```
128
129     String query = String.format("%s where %s %s",
getAllQuery(), whereQuery, limitQuery);
130     PreparedStatement stmt = conn.prepareStatement(query);
131     ResultSet r = stmt.executeQuery();
132
133     while (r.next())
134     {
135         T i = createBean(r);
136         if (!result.contains(i)) {
137             result.add(i);
138         }
139     }
140     ConnectionFactory.closeConn(r, stmt, conn);
141     return result;
142 }
143
144 public T findOne(String val) throws Exception {
145     return this.findOneBy("id", val, false);
146 }
147
148 public T findOneBy(String by, String val, Boolean like) throws
Exception {
149     return this.findOneBy(by, val, like, 0);
150 }
151
152 public T findOneBy(String by, String val, Boolean like,
Integer limit) throws Exception {
153     return this.getAllBy(by, val, like, limit).get(0);
154 }
155
156 public abstract Map<String, String[]> getColumns();
157 public abstract String convertToColumnName(String by);
158 public abstract List<String[]> getSearchColumns();
159 public abstract T createBean(ResultSet r) throws Exception;
160 }
161
```

CategoryDAO.java

```
1 package model.dao;
2
3 import java.sql.Blob;
4
11
12 public class CategoryDAO extends DAO<Category> {
13     private static Map<String, String[]> COLUMNS;
14     private static List<String[]> searchColumns;
15
16     public CategoryDAO() throws Exception {
17         super("CATEGORY");
18         COLUMNS = new HashMap<String, String[]>();
19         searchColumns = new ArrayList<String[]>();
20
21         COLUMNS.put("name", new String[]{"NAME", "str"});
22         COLUMNS.put("desc", new String[]{"DESCRIPTION", "str"});
23         COLUMNS.put("pic", new String[]{"PICTURE", "str"});
24         COLUMNS.put("id", new String[]{"ID", "num"});
25
26         searchColumns.add(COLUMNS.get("name"));
27         searchColumns.add(COLUMNS.get("desc"));
28     }
29
30     @Override
31     public String convertToColumnName(String by) {
32         String result = COLUMNS.get("id")[0];
33
34         if (COLUMNS.containsKey(by) ||
35             COLUMNS.containsKey(by.toLowerCase()))
36         {
37             result = COLUMNS.get(by)[0];
38         }
39         else if (by.equals(COLUMNS.get("name")[0]))
40         {
41             result = COLUMNS.get("name")[0];
42         }
43         else if (by.toLowerCase().equals("description") ||
44             by.equals(COLUMNS.get("desc")[0]))
45         {
46             result = COLUMNS.get("desc")[0];
47         }
48         else if (by.toLowerCase().equals("picture") ||
49             by.equals(COLUMNS.get("pic")[0]))
50         {
51             result = COLUMNS.get("pic")[0];
52         }
53     }
54 }
```

CategoryDAO.java

```
49     }
50     else if (by.equals(COLUMNS.get("id")[0]))
51     {
52         result = COLUMNS.get("id")[0];
53     }
54     return result;
55 }
56
57 @Override
58 public List<String[]> getSearchColumns() {
59     return searchColumns;
60 }
61
62 @Override
63 public Map<String, String[]> getColumns() {
64     return COLUMNS;
65 }
66
67 @Override
68 public Category createBean(ResultSet r) throws Exception {
69     Category result = new Category();
70     result.setId(getId(r));
71     result.setName(getName(r));
72     result.setDescription(getDescription(r));
73     result.setPicture(getPicture(r));
74     return result;
75 }
76
77 public String getName(ResultSet r) throws Exception {
78     return r.getString(COLUMNS.get("name")[0]);
79 }
80
81 public String getDescription(ResultSet r) throws Exception {
82     return r.getString(COLUMNS.get("desc")[0]);
83 }
84
85 public int getId(ResultSet r) throws Exception {
86     return r.getInt(COLUMNS.get("id")[0]);
87 }
88
89 public Blob getPicture(ResultSet r) throws Exception {
90     return r.getBlob(COLUMNS.get("pic")[0]);
91 }
92
```

CategoryDAO.java

```
93     public static void main(String[] args) throws Exception {
94         CategoryDAO catDao = new CategoryDAO();
95         System.out.println(catDao.getAll().toString());
96     }
97 }
98
```


ItemDAO.java

```

1 package model.dao;
2
3 import java.sql.ResultSet;
4
10
11 public class ItemDAO extends DAO<Item> {
12     private static Map<String, String[]> COLUMNS;
13     private static List<String[]> searchColumns;
14
15     public ItemDAO() throws Exception {
16         super("ITEM");
17         COLUMNS = new HashMap<String, String[]>();
18         searchColumns = new ArrayList<String[]>();
19
20         COLUMNS.put("qty", new String[]{"QTY", "num"});
21         COLUMNS.put("name", new String[]{"NAME", "str"});
22         COLUMNS.put("price", new String[]{"PRICE", "num"});
23         COLUMNS.put("id", new String[]{"NUMBER", "str"});
24         COLUMNS.put("num", new String[]{"NUMBER", "str"});
25         COLUMNS.put("cid", new String[]{"CATID", "num"});
26         COLUMNS.put("unit", new String[]{"UNIT", "str"});
27
28         searchColumns.add(COLUMNS.get("name"));
29         searchColumns.add(COLUMNS.get("num"));
30     }
31
32     @Override
33     public String convertToColumnName(String by) {
34         String result = COLUMNS.get("cid")[0];
35
36         if (COLUMNS.containsKey(by) ||
37 COLUMNS.containsKey(by.toLowerCase()))
38         {
39             result = COLUMNS.get(by)[0];
40         }
41         else if (by.equals(COLUMNS.get("name")[0]))
42         {
43             result = COLUMNS.get("name")[0];
44         }
45         else if (by.toLowerCase().equals("price") ||
46 by.equals(COLUMNS.get("price")[0]))
47         {
48             result = COLUMNS.get("price")[0];
49         }
50         else if (by.toLowerCase().equals("quantity") ||

```

ItemDAO.java

```
by.equals(COLUMNS.get("qty")[0]))
49     {
50         result = COLUMNS.get("qty")[0];
51     }
52     else if (by.toLowerCase().equals("number") ||
by.toLowerCase().equals("num") || by.equals(COLUMNS.get("id")[0]))
53     {
54         result = COLUMNS.get("id")[0];
55     }
56     else if (by.toLowerCase().equals(COLUMNS.get("cid")[0]))
57     {
58         result = COLUMNS.get("cid")[0];
59     }
60     else if (by.toLowerCase().equals(COLUMNS.get("unit")[0]))
61     {
62         result = COLUMNS.get("unit")[0];
63     }
64     return result;
65 }
66
67 @Override
68 public List<String[]> getSearchColumns() {
69     return searchColumns;
70 }
71
72 @Override
73 public Item createBean(ResultSet r) throws Exception {
74     Item item = new Item();
75     item.setName(getName(r));
76     item.setQuantity(getQuantity(r));
77     item.setPrice(getPrice(r));
78     item.setNumber(getNumber(r));
79     item.setCatID(getCatID(r));
80     item.setUnit(getUnit(r));
81     return item;
82 }
83
84 @Override
85 public Map<String, String[]> getColumns() {
86     return COLUMNS;
87 }
88
89 public String getId(ResultSet r) throws Exception {
90     return r.getString(COLUMNS.get("id")[0]);
```

ItemDAO.java

```
91     }
92
93     public String getName(ResultSet r) throws Exception {
94         return r.getString(COLUMNS.get("name")[0]);
95     }
96
97     public double getPrice(ResultSet r) throws Exception {
98         return r.getDouble(COLUMNS.get("price")[0]);
99     }
100
101     public int getQuantity(ResultSet r) throws Exception {
102         return r.getInt(COLUMNS.get("qty")[0]);
103     }
104
105     public String getNumber(ResultSet r) throws Exception {
106         return r.getString(COLUMNS.get("num")[0]);
107     }
108
109     public int getCatID(ResultSet r) throws Exception {
110         return r.getInt(COLUMNS.get("cid")[0]);
111     }
112
113     public String getUnit(ResultSet r) throws Exception {
114         return r.getString(COLUMNS.get("unit")[0]);
115     }
116
117     public static void main(String[] args) throws Exception {
118         ItemDAO itemDao = new ItemDAO();
119         System.out.println(itemDao.getAll().toString());
120     }
121 }
122
```

OrderDAO.java

```
1 package model.dao;
2
3 import java.io.File;
4
5
6
7
8 public class OrderDAO {
9
10     private File ordersDir;
11
12     public OrderDAO() {
13     }
14
15     public OrderDAO(String ordersDir) {
16         this(new File(ordersDir));
17     }
18
19     public OrderDAO(File ordersDir) {
20         this.setOrdersDir(ordersDir);
21         checkDir();
22     }
23
24     public synchronized String getOrderFileName(String accountNum,
25 int id) {
26         String result = "";
27         if (id > 0) {
28             result = getOrderFileNameFirstPart(accountNum) + id +
29 ".xml";
30         }
31         return result;
32     }
33
34     public synchronized String getOrderFileNameFirstPart(String
35 accountNum) {
36         return "po" + accountNum + "_";
37     }
38
39     public synchronized File[] getP0s() {
40         return ordersDir.listFiles();
41     }
42
43     public synchronized File[] getP0s(String accountNum) {
44         return ordersDir.listFiles(new FilenameFilter() {
45             @Override public boolean accept(File dir, String name)
46         }
47         return
```

OrderDAO.java

```
    name.startsWith(getOrderFileNameFirstPart(accountNum));
44        }
45    });
46    }
47
48    public synchronized File getP0(String fileName) {
49        return new File(ordersDir, fileName);
50    }
51
52    public synchronized FileWriter getFileWriter(String fileName)
    throws IOException {
53        File poFile = getP0(fileName);
54        if (poFile.exists()) {
55            throw new IOException("File already exists.");
56        }
57        poFile.createNewFile();
58        return new FileWriter(poFile);
59    }
60
61    /**
62     * Check if directory exists, else create a new one
63     */
64    public synchronized void checkDir() {
65        if (!ordersDir.exists()) {
66            ordersDir.mkdir();
67        }
68    }
69
70    public File getOrdersDir() {
71        return this.ordersDir;
72    }
73
74    public void setOrdersDir(File dir) {
75        this.ordersDir = dir;
76    }
77
78 }
79
```



Model.Helpers

Utils.java

```
1 package model.helpers;
2
3 import java.io.BufferedReader;
40
41 public class Utils {
42
43     public static final String AUTH_URL = "https://
www.eecs.yorku.ca/~roumani/servers/auth/oauth.cgi";
44
45     public static String doHttpGet(Map<String, String> params,
String url, String contentType) throws Exception
46     {
47         StringBuilder result = new StringBuilder();
48         for (Map.Entry<String, String> entry: params.entrySet()) {
49             result.append(URLEncoder.encode(entry.getKey(),
"UTF-8"));
50             result.append("=");
51             result.append(URLEncoder.encode(entry.getValue(),
"UTF-8"));
52             result.append("&");
53         }
54         String parameters = result.toString();
55
56         URL url = new URL(url + "?" + parameters);
57         HttpURLConnection con = (HttpURLConnection)
url.openConnection();
58         con.setRequestMethod("GET");
59         con.setRequestProperty("Content-Type", contentType);
60         con.setDoOutput(true);
61
62         BufferedReader in = new BufferedReader(
63             new InputStreamReader(con.getInputStream()));
64
65         String inputLine;
66         StringBuffer response = new StringBuffer();
67
68         while ((inputLine = in.readLine()) != null) {
69             response.append(inputLine);
70         }
71         in.close();
72         con.disconnect();
73
74         return response.toString();
75     }
}
```

Utils.java

```
76
77     public static void doAuth(String username, String pass, String
    back) throws Exception {
78         HttpURLConnection connection = (HttpURLConnection) new
    URL(AUTH_URL + "?back=" + back).openConnection();
79         String encoded =
    Base64.getEncoder().encodeToString((username+":"+pass).getBytes(St
    andardCharsets.UTF_8)); //Java 8
80         connection.setRequestProperty("Authorization", "Basic
    "+encoded);
81     }
82
83     public static void renderXsltTHtml(String xml, InputStream
    xslIs, Writer out) throws Exception {
84         DocumentBuilder factory_db =
    DocumentBuilderFactory.newInstance().newDocumentBuilder();
85         Document doc = factory_db.parse(new InputSource(new
    StringReader(xml)));
86         Source xmlSource = new DOMSource(doc);
87
88         TransformerFactory transformerFactory =
    TransformerFactory.newInstance();
89         Templates cachedXSLT = transformerFactory.newTemplates(new
    StreamSource(xslIs));
90         Transformer transformer = cachedXSLT.newTransformer();
91
92         transformer.transform(xmlSource, new StreamResult(out));
93     }
94
95     public static double round(double value, int places) {
96         if (places < 0) throw new IllegalArgumentException();
97
98         BigDecimal bd = new BigDecimal(value);
99         bd = bd.setScale(places, RoundingMode.HALF_UP);
100        return bd.doubleValue();
101    }
102
103    public static Boolean isValidUser(Account user) {
104        boolean result = false;
105        if (user != null && user.getName() != null && !
    user.getName().isEmpty()
106            && user.getUsername() != null && !
    user.getUsername().isEmpty()) {
107            result = true;
```


Utils.java

```
108     }  
109     return result;  
110 }  
111  
112 }  
113
```



Controller

Home.java

```
1 package ctrl;
2
3 import java.io.IOException;
4
5
6
7
8
9
10
11
12 /**
13  * Servlet implementation class Home
14  */
15 @WebServlet("/Home")
16 public class Home extends HttpServlet {
17     private static final long serialVersionUID = 1L;
18
19     /**
20      * @see HttpServlet#HttpServlet()
21      */
22     public Home() {
23         super();
24         // TODO Auto-generated constructor stub
25     }
26
27     /**
28      * @see HttpServlet#doGet(HttpServletRequest request,
29      * HttpServletResponse response)
30      */
31     protected void doGet(HttpServletRequest request,
32     HttpServletResponse response) throws ServletException, IOException
33     {
34         // TODO Auto-generated method stub
35         String getFoodByCat = request.getParameter("getFoodByCat");
36
37         Model model = Model.getInstance();
38
39         if (getFoodByCat == null) {
40             try {
41                 request.setAttribute("catsWithFoods",
42                 model.getCatsWithFoods());
43                 request.setAttribute("cats",
44                 model.getCategories());
45             } catch (Exception e) {
46                 request.setAttribute("error", e.getMessage());
47             }
48         }
49     }
50 }
```

Home.java

```
46         request.getRequestDispatcher("/WEB-INF/pages/  
Home.jspx").forward(request, response);  
47     }  
48  
49     /**  
50     * @see HttpServlet#doPost(HttpServletRequest request,  
HttpServletResponse response)  
51     */  
52     protected void doPost(HttpServletRequest request,  
HttpServletResponse response) throws ServletException, IOException  
    {  
53         // TODO Auto-generated method stub  
54         doGet(request, response);  
55     }  
56  
57 }  
58
```

Cart.java

```
1 package ctrl;
2
3 import java.io.IOException;
4
5
6
7
8
9
10
11
12
13
14
15
16
17 /**
18  * Servlet implementation class Cart
19  */
20 @WebServlet("/Cart")
21 public class Cart extends HttpServlet {
22     private static final long serialVersionUID = 1L;
23
24     /**
25      * @see HttpServlet#HttpServlet()
26      */
27     public Cart() {
28         super();
29         // TODO Auto-generated constructor stub
30     }
31
32     /**
33      * @see HttpServlet#doGet(HttpServletRequest request,
34      HttpServletResponse response)
35      */
36     protected void doGet(HttpServletRequest request,
37 HttpServletResponse response) throws ServletException, IOException
38     {
39         HttpSession session = request.getSession();
40         request.setAttribute("order",
41 session.getAttribute("order"));
42         request.getRequestDispatcher("/WEB-INF/pages/
43 Cart.jsp").forward(request, response);
44     }
45
46     /**
47      * @see HttpServlet#doPost(HttpServletRequest request,
48 HttpServletResponse response) throws ServletException, IOException
49      */
50     protected void doPost(HttpServletRequest request,
51 HttpServletResponse response) throws ServletException, IOException
52     {
53         // TODO Auto-generated method stub
54         doGet(request, response);
55     }
56 }
57 }
```

Checkout.java

```
1 package ctrl;
2
3 import java.io.File;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 /**
22  * Servlet implementation class Checkout
23  */
24 @WebServlet("/Checkout")
25 public class Checkout extends HttpServlet {
26     private static final long serialVersionUID = 1L;
27
28     /**
29      * @see HttpServlet#HttpServlet()
30      */
31     public Checkout() {
32         super();
33         // TODO Auto-generated constructor stub
34     }
35
36     /**
37      * @see HttpServlet#doGet(HttpServletRequest request,
38      HttpServletResponse response)
39      */
40     protected void doGet(HttpServletRequest request,
41 HttpServletResponse response) throws ServletException, IOException
42     {
43         HttpSession s = request.getSession();
44         Model m = Model.getInstance();
45         Account user = (Account) s.getAttribute("AUTH");
46         Order order = (Order) s.getAttribute("order");
47
48         ServletContext context = getServletContext();
49         String poPath = context.getRealPath("/POs");
50
51         String confirm = request.getParameter("confirm");
52         String cancel = request.getParameter("cancel");
53         if (cancel != null) {
54             String redirectUrl = "Cart";
55             response.sendRedirect(redirectUrl);
56         }
57
58         if (confirm != null) {
59             if (order.getItems().size() > 0 &&
60 Utils.isValidUser(user)) {
```

Checkout.java

```
57         order.setCustomer(user);
58         order.setSubmitted(new Date());
59         order.setId(((Integer)
request.getAttribute("totalAllOrders")));
60
61         try {
62             int orderNum = ((Integer)
s.getAttribute("totalOrders")) + 1;
63             m.createPO(poPath, orderNum, "res/xml/P0.xml",
order, user);
64
65             order.clearCart();
66             request.setAttribute("orderCreated", orderNum);
67         } catch (Exception e) {
68             request.setAttribute("error", e.getMessage());
69             System.out.println("ERROR: " + e.getMessage());
70         }
71     } else {
72         request.setAttribute("error", "Empty Cart!");
73         System.out.println("Empty Cart!");
74     }
75 }
76
77     request.setAttribute("order", s.getAttribute("order"));
78
79     request.getRequestDispatcher("/WEB-INF/pages/
Checkout.jspx").forward(request, response);
80 }
81
82 /**
83  * @see HttpServlet#doPost(HttpServletRequest request,
HttpServletResponse response)
84  */
85     protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException
{
86         // TODO Auto-generated method stub
87         doGet(request, response);
88     }
89
90 }
91
```

PurchaseOrders.java

```
1 package ctrl;
2
3 import java.io.File;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25 /**
26  * Servlet implementation class PurchaseOrders
27  */
28 @WebServlet("/PurchaseOrders")
29 public class PurchaseOrders extends HttpServlet {
30     private static final long serialVersionUID = 1L;
31
32     /**
33      * @see HttpServlet#HttpServlet()
34      */
35     public PurchaseOrders() {
36         super();
37         // TODO Auto-generated constructor stub
38     }
39
40     /**
41      * @see HttpServlet#doGet(HttpServletRequest request,
42      * HttpServletResponse response)
43      */
44     protected void doGet(HttpServletRequest request,
45     HttpServletResponse response) throws ServletException, IOException
46     {
47         // response.setContentType("text/xml");
48
49         String poPath = getServletContext().getRealPath("/POs");
50         String xslFilePath = getServletContext().getRealPath("res/
51         xml/P0.xsl");
52
53         HttpSession s = request.getSession();
54         Writer out = response.getWriter();
55         StringWriter sw = null;
56
57         String orderNum = request.getParameter("orderNum");
58         Account user = (Account) s.getAttribute("AUTH");
59
60         OrderDAO orderDao = new OrderDAO(new File(poPath));
61         File[] orders = orderDao.getPOs(user.getUsername());
62         File orderFile = null;
63
64         String[] userOrderLinks = new String[orders.length + 1];
```


PurchaseOrders.java

```
61     Map<String, String> userOrderLinks = new HashMap<>();
62
63     try {
64         if (orders.length > 0) {
65             int i=1;
66             for (File f: orders) {
67                 String fileName =
68 f.getAbsolutePath().replace(poPath, "");
69                 fileName = fileName.replaceAll("/", "");
70                 fileName = fileName.replaceAll(".xml", "");
71                 fileName.replace(orderDao.getOrderFileNameFirstPart(user.getUserName()), "");
72                 userOrderLinks.put(fileName + "",
73 request.getContextPath() + "/PurchaseOrders?orderNum=" +
74 fileName);
75
76                 if (orderNum != null &&
77 fileName.equals(orderNum)) {
78                     orderFile = f;
79                 }
80                 i++;
81             }
82
83             if (orderFile != null) {
84                 sw = new StringWriter();
85                 Source xml = new StreamSource(orderFile);
86                 Source xslt = new StreamSource(xsltFilePath);
87
88                 TransformerFactory tFactory =
89 TransformerFactory.newInstance();
90                 Transformer transform =
91 tFactory.newTransformer(xslt);
92                 transform.transform(xml, new StreamResult(sw));
93             } else {
94                 if (orderNum == null) {
95                     s.setAttribute("ordersLinks",
96 userOrderLinks);
97                 }
98                 request.setAttribute("ordersLinks",
99 userOrderLinks);
100             } else {
101                 request.setAttribute("error", "No Purchase
102 Order found!");
103             }
104         }
105     } catch (Exception e) {
106         // Handle exception
107     }
108 }
```

PurchaseOrders.java

```
94         }
95     }
96     } else {
97         request.setAttribute("error", "No Purchase Order
for current user!");
98     }
99
100     } catch (Exception e) {
101         request.setAttribute("error", e.getMessage());
102     }
103
104     if (sw != null) {
105         out.write(sw.toString());
106     } else {
107         request.getRequestDispatcher("/WEB-INF/pages/
PurchaseOrders.jspx").forward(request, response);
108     }
109 }
110
111 /**
112  * @see HttpServlet#doPost(HttpServletRequest request,
HttpServletResponse response)
113  */
114 protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException
{
115     // TODO Auto-generated method stub
116     doGet(request, response);
117 }
118
119 }
120
```

Admin.java

```
1 package ctrl;
2
3 import java.io.IOException;
13
14 /**
15  * Servlet implementation class Admin
16  */
17 @WebServlet("/Admin")
18 public class Admin extends HttpServlet {
19     private static final long serialVersionUID = 1L;
20
21     /**
22      * @see HttpServlet#HttpServlet()
23      */
24     public Admin() {
25         super();
26         // TODO Auto-generated constructor stub
27     }
28
29     /**
30      * @see HttpServlet#doGet(HttpServletRequest request,
31      *      HttpServletResponse response)
32      */
33     protected void doGet(HttpServletRequest request,
34     HttpServletResponse response) throws ServletException, IOException
35     {
36         // TODO Auto-generated method stub
37         Account user = (Account)
38         request.getSession().getAttribute("AUTH");
39
40         if (Utils.isValidUser(user)) {
41             request.setAttribute("addToCartAvgCount",
42             request.getSession().getAttribute("addToCartAvgCount"));
43             request.setAttribute("addToCartAvgTime",
44             request.getSession().getAttribute("addToCartAvgTime"));
45
46             request.setAttribute("checkoutAvgTime",
47             request.getSession().getAttribute("checkoutAvgTime"));
48             request.setAttribute("checkoutAvgCount",
49             request.getSession().getAttribute("checkoutAvgCount"));
50         } else {
51             request.setAttribute("error", "Invalid User!");
52         }
53     }
54 }
```

Admin.java

```
46         request.getRequestDispatcher("/WEB-INF/pages/  
Admin.jspx").forward(request, response);  
47     }  
48  
49     /**  
50     * @see HttpServlet#doPost(HttpServletRequest request,  
HttpServletResponse response)  
51     */  
52     protected void doPost(HttpServletRequest request,  
HttpServletResponse response) throws ServletException, IOException  
    {  
53         // TODO Auto-generated method stub  
54         doGet(request, response);  
55     }  
56  
57 }  
58
```



Controller.Api

Product.java

```
1 package ctrl.api;
2
3 import java.io.IOException;
23
24 /**
25  * Servlet implementation class Product
26  */
27 @WebServlet("/Api/Product")
28 public class Product extends HttpServlet {
29     private static final long serialVersionUID = 1L;
30
31     /**
32      * @see HttpServlet#HttpServlet()
33      */
34     public Product() {
35         super();
36         // TODO Auto-generated constructor stub
37     }
38
39     /**
40      * @see HttpServlet#doGet(HttpServletRequest request,
41      *      HttpServletResponse response)
42      */
43     protected void doGet(HttpServletRequest request,
44     HttpServletResponse response) throws ServletException, IOException
45     {
46         response.setContentType("application/json");
47         response.setCharacterEncoding("utf-8");
48         response.addHeader("Access-Control-Allow-Origin", "*");
49
50         PrintWriter out = response.getWriter();
51         JsonObject json = new JsonObject();
52         JsonElement resultJson;
53         Model model = Model.getInstance();
54
55         String num = request.getParameter("num");
56         String cid = request.getParameter("cid");
57         String limit = request.getParameter("limit");
58         String byCat = request.getParameter("byCat");
59
60         try {
61             if (byCat != null) {
62                 if (byCat.toLowerCase().equals("name")) {
63                     Map<String, List<Item>> result =
```

Product.java

```
        model.getCatNameWithFoods();
61         resultJson = new Gson().toJsonTree(result);
62     } else {
63         Map<Category, List<Item>> result =
        model.getCatsWithFoods();
64
65         resultJson = new Gson().toJsonTree(result);
66     }
67     } else {
68         List<Item> result = model.getFoods();
69
70         if (num != null)
71         {
72             result = model.getFoodsBy("num", num, false);
73         }
74         else if (cid != null)
75         {
76             result = model.getFoodsBy("cid", cid, false);
77         }
78
79         if (limit != null) {
80             int l = Integer.parseInt(limit);
81             result = result.subList(0, l);
82         }
83         resultJson = new Gson().toJsonTree(result);
84     }
85
86     json.add("data", resultJson);
87     json.addProperty("status", 1);
88 } catch (Exception e) {
89     json.addProperty("status", 0);
90     json.addProperty("data", e.getMessage());
91 }
92
93     out.print(json.toString());
94 }
95
96 /**
97  * @see HttpServlet#doPost(HttpServletRequest request,
    HttpServletResponse response)
98  */
99     protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException
    {
```

Product.java

```
100      // TODO Auto-generated method stub
101      doGet(request, response);
102  }
103
104 }
105
```


CartApi.java

```
1 package ctrl.api;
2
3 import java.io.IOException;
4
5 /**
6  * Servlet implementation class CartApi
7  */
8 @WebServlet("/Api/Cart")
9 public class CartApi extends HttpServlet {
10     private static final long serialVersionUID = 1L;
11
12     /**
13      * @see HttpServlet#HttpServlet()
14      */
15     public CartApi() {
16         super();
17         // TODO Auto-generated constructor stub
18     }
19
20     /**
21      * @see HttpServlet#doGet(HttpServletRequest request,
22      HttpServletResponse response)
23      */
24     protected void doGet(HttpServletRequest request,
25 HttpServletResponse response) throws ServletException, IOException
26     {
27         response.setContentType("application/json");
28         response.setCharacterEncoding("utf-8");
29         response.addHeader("Access-Control-Allow-Origin", "*");
30
31         PrintWriter out = response.getWriter();
32         JsonObject json = new JsonObject();
33         JsonElement resultJson;
34         Model model = Model.getInstance();
35         HttpSession session = request.getSession();
36
37         Order userOrder = (Order) session.getAttribute("order");
38         List<Item> products = userOrder.getItems();
39
40         String addToCart = request.getParameter("addToCart");
41         String removeItem = request.getParameter("removeItem");
42         String updateQty = request.getParameter("updateQty");
43         String productQty = request.getParameter("qty");
44     }
45 }
```

CartApi.java

```
62         json.addProperty("status", 1);
63
64         if (addToCart != null) {
65             addItemToCart(model, products, json, addToCart,
productQty);
66         }
67
68         if (updateQty != null) {
69             updateQty(products, json, updateQty, productQty);
70         }
71
72         if (removeItem != null) {
73             removeItem(products, json, removeItem);
74         }
75
76         resultJson = new Gson().toJsonTree(userOrder);
77         if (!json.has("data")) {
78             json.add("data", resultJson);
79         }
80
81         if (!json.has("action")) {
82             json.addProperty("action", "GET");
83         }
84
85         out.print(json.toString());
86     }
87
88     /**
89      * @see HttpServlet#doPost(HttpServletRequest request,
HttpServletResponse response)
90      */
91     protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException
92     {
93         // TODO Auto-generated method stub
94         doGet(request, response);
95     }
96
97     private void addItemToCart(Model model, List<Item> products,
JsonObject json, String addToCart, String productQty) {
98         try {
99             Item product = model.getFoodBy("num", addToCart);
100             if (products.size() > 0) {
101                 if (!products.contains(product)) {
```

CartApi.java

```
101         if (productQty != null && !
productQty.isEmpty()) {
102             product.setQuantity(productQty);
103         }
104
105         products.add(product);
106
107         json.addProperty("status", 1);
108         json.addProperty("action", "ADD");
109         json.addProperty("actionReq", addToCart);
110     } else {
111         // int i = products.indexOf(product);
112         // Item p = products.get(i);
113         // if (productQty != null && !
productQty.isEmpty()) {
114         //     p.increaseQty(productQty);
115         // }
116     }
117     } else {
118         if (productQty != null && !productQty.isEmpty()) {
119             product.setQuantity(productQty);
120         }
121
122         products.add(product);
123         json.addProperty("status", 1);
124
125         json.addProperty("action", "ADD");
126         json.addProperty("actionReq", addToCart);
127     }
128 } catch (Exception e) {
129     json.addProperty("status", 0);
130     json.addProperty("data", e.getMessage());
131 }
132 }
133
134
135 private void updateQty(List<Item> products, JsonObject json,
String updateQty, String productQty) {
136     try {
137         Item product;
138         for (Item it: products) {
139             if (it.getNumber().equals(updateQty)) {
140                 if (productQty != null) {
141                     if (productQty.equals("0")) {
```

CartApi.java

```
142         products.remove(it);
143     } else {
144         product = it;
145         product.setQuantity(productQty);
146     }
147     json.addProperty("status", 1);
148 } else {
149     json.addProperty("status", 0);
150     json.addProperty("data", "No Quantity
Provided.");
151 }
152 json.addProperty("action", "UPDATE");
153 json.addProperty("actionReq", updateQty);
154 }
155 }
156 } catch (Exception e) {
157     json.addProperty("status", 0);
158     json.addProperty("data", e.getMessage());
159 }
160 }
161
162 private void removeItem(List<Item> products, JsonObject json,
String removeItem) {
163     try {
164         for (Item it: products) {
165             if (it.getNumber().equals(removeItem)) {
166                 products.remove(it);
167
168                 json.addProperty("status", 1);
169                 json.addProperty("action", "REMOVE");
170                 json.addProperty("actionReq", removeItem);
171             }
172         }
173     } catch (Exception e) {
174         json.addProperty("status", 0);
175         json.addProperty("data", e.getMessage());
176     }
177 }
178
179 }
180
```



Analytics

Monitor.java

```
1 package analytics;
2
3 import java.util.ArrayList;
4
5
6
7
8
9
10
11
12
13
14 /**
15  * Application Lifecycle Listener implementation class Monitor
16  *
17  */
18 @WebListener
19 public class Monitor implements ServletContextListener,
20     ServletContextAttributeListener, HttpSessionListener,
21     HttpSessionAttributeListener, HttpSessionActivationListener,
22     HttpSessionBindingListener, HttpSessionIdListener,
23     ServletRequestListener, ServletRequestAttributeListener,
24     AsyncListener {
25
26     /**
27      * Default constructor.
28      */
29     public Monitor() {
30         // TODO Auto-generated constructor stub
31     }
32
33     /**
34      * @see HttpSessionListener#sessionCreated(HttpSessionEvent)
35      */
36     public void sessionCreated(HttpSessionEvent se) {
37         // TODO Auto-generated method stub
38         List<Item> items = new ArrayList<Item>();
39         se.getSession().setAttribute("order", new Order(items));
40         se.getSession().setAttribute("orders", new
41             ArrayList<String>());
42         se.getSession().setAttribute("AUTH", new Account());
43         se.getSession().setAttribute("startTime", new Date());
44     }
45
46     /**
47      * @see
48      ServletContextAttributeListener#attributeRemoved(ServletContextAtt
49      ributeEvent)
50      */
51     public void attributeRemoved(ServletContextAttributeEvent
52         scae) {
53         // TODO Auto-generated method stub
54     }
55 }
```

Monitor.java

```
65     }
66
67     /**
68      * @see AsyncListener#onError(AsyncEvent)
69      */
70     public void onError(AsyncEvent arg0) throws
java.io.IOException {
71         // TODO Auto-generated method stub
72     }
73
74     /**
75      * @see
76      * HttpSessionIdListener#sessionIdChanged(HttpSessionEvent, String)
77      */
77     public void sessionIdChanged(HttpSessionEvent arg0, String
arg1) {
78         // TODO Auto-generated method stub
79         if (arg0.getSession().getAttribute("orders") == null) {
80             arg0.getSession().setAttribute("orders", new
ArrayList<String>());
81         }
82
83         if (arg0.getSession().getAttribute("order") == null) {
84             List<Item> items = new ArrayList<Item>();
85             arg0.getSession().setAttribute("order", new Order(items));
86         }
87
88         if (arg0.getSession().getAttribute("AUTH") == null) {
89             arg0.getSession().setAttribute("AUTH", new Account());
90         }
91
92         if (arg0.getSession().getAttribute("startTime") == null) {
93             arg0.getSession().setAttribute("startTime", new Date());
94         }
95     }
96
97     /**
98      * @see
99      * ServletRequestAttributeListener#attributeAdded(ServletRequestAttri
100      * buteEvent)
101     */
100     public void attributeAdded(ServletRequestAttributeEvent srae)
{
101         // TODO Auto-generated method stub
```

Monitor.java

```
102     }
103
104     /**
105      * @see AsyncListener#onTimeout(AsyncEvent)
106      */
107     public void onTimeout(AsyncEvent arg0) throws
108     java.io.IOException {
109         // TODO Auto-generated method stub
110     }
111
112     /**
113      * @see
114      HttpSessionAttributeListener#attributeReplaced(HttpSessionBindingE
115      vent)
116      */
117     public void attributeReplaced(HttpSessionBindingEvent se) {
118         // TODO Auto-generated method stub
119     }
120
121     /**
122      * @see
123      HttpSessionActivationListener#sessionWillPassivate(HttpSessionEven
124      t)
125      */
126     public void sessionWillPassivate(HttpSessionEvent se) {
127         // TODO Auto-generated method stub
128     }
129
130     /**
131      * @see
132      ServletContextListener#contextInitialized(ServletContextEvent)
133      */
134     public void contextInitialized(ServletContextEvent sce) {
135         // TODO Auto-generated method stub
136     }
137
138     /**
139      * @see
140      ServletContextAttributeListener#attributeAdded(ServletContextAttri
141      buteEvent)
142      */
143     public void attributeAdded(ServletContextAttributeEvent event)
144     {
145         // TODO Auto-generated method stub
146     }
147 }
```


Monitor.java

```
137     }
138
139     /**
140      * @see AsyncListener#onComplete(AsyncEvent)
141      */
142     public void onComplete(AsyncEvent event) throws
java.io.IOException {
143         // TODO Auto-generated method stub
144     }
145
146     /**
147      * @see
ServletRequestListener#requestDestroyed(ServletRequestEvent)
148      */
149     public void requestDestroyed(ServletRequestEvent sre) {
150         // TODO Auto-generated method stub
151     }
152
153     /**
154      * @see
ServletRequestAttributeListener#attributeRemoved(ServletRequestAtt
ributeEvent)
155      */
156     public void attributeRemoved(ServletRequestAttributeEvent
srae) {
157         // TODO Auto-generated method stub
158     }
159
160     /**
161      * @see AsyncListener#onStartAsync(AsyncEvent)
162      */
163     public void onStartAsync(AsyncEvent event) throws
java.io.IOException {
164         // TODO Auto-generated method stub
165     }
166
167     /**
168      * @see
HttpSessionBindingListener#valueBound(HttpSessionBindingEvent)
169      */
170     public void valueBound(HttpSessionBindingEvent event) {
171         // TODO Auto-generated method stub
172     }
173
```

Monitor.java

```
174     /**
175      * @see
ServletRequestListener#requestInitialized(ServletRequestEvent)
176      */
177     public void requestInitialized(ServletRequestEvent sre) {
178         HttpServletRequest req = (HttpServletRequest)
sre.getServletRequest();
179         HttpSession currentSession = req.getSession();
180
181         if (req.getAttribute("users") == null) {
182             req.setAttribute("users", new HashMap<String, long[]>());
183         }
184
185         if (req.getAttribute("poDir") == null) {
186             req.setAttribute("poDir",
req.getServletContext().getRealPath("/POs"));
187         }
188
189         if (currentSession.getAttribute("order") == null) {
190             List<Item> items = new ArrayList<Item>();
191             currentSession.setAttribute("order", new Order(items));
192         }
193
194         if (currentSession.getAttribute("AUTH") == null) {
195             currentSession.setAttribute("AUTH", new Account());
196         }
197
198         if (currentSession.getAttribute("orders") == null) {
199             currentSession.setAttribute("orders", new
ArrayList<String>());
200         }
201
202         ComputeAnalytics.computeUsersAnalytics(req);
203         ComputeAnalytics.computeOrders(req);
204         ComputeAnalytics.addUserOrders(req);
205         ComputeAnalytics.computeAvgStartCheckoutTime(req);
206         ComputeAnalytics.computeAvgAddToCartTime(req);
207     }
208
209     /**
210      * @see HttpSessionListener#sessionDestroyed(HttpSessionEvent)
211      */
212     public void sessionDestroyed(HttpSessionEvent se) {
213         // TODO Auto-generated method stub
```

Monitor.java

```
214     }
215
216     /**
217     * @see
218     HttpSessionActivationListener#sessionDidActivate(HttpSessionEvent)
219     */
219     public void sessionDidActivate(HttpSessionEvent se) {
220         // TODO Auto-generated method stub
221     }
222
223     /**
224     * @see
225     ServletContextListener#contextDestroyed(ServletContextEvent)
226     */
226     public void contextDestroyed(ServletContextEvent sce) {
227         // TODO Auto-generated method stub
228     }
229
230     /**
231     * @see
232     ServletRequestAttributeListener#attributeReplaced(ServletRequestAttributeEvent)
233     */
233     public void attributeReplaced(ServletRequestAttributeEvent srae) {
234         // TODO Auto-generated method stub
235     }
236
237     /**
238     * @see
239     HttpSessionAttributeListener#attributeAdded(HttpSessionBindingEvent)
240     */
240     public void attributeAdded(HttpSessionBindingEvent event) {
241         // TODO Auto-generated method stub
242     }
243
244     /**
245     * @see
246     HttpSessionAttributeListener#attributeRemoved(HttpSessionBindingEvent)
247     */
247     public void attributeRemoved(HttpSessionBindingEvent event) {
248         // TODO Auto-generated method stub
```

Monitor.java

```
249     }
250
251     /**
252     * @see
253     ServletContextAttributeListener#attributeReplaced(ServletContextAt
254     tributeEvent)
255     */
256     public void attributeReplaced(ServletContextAttributeEvent
257     event) {
258         // TODO Auto-generated method stub
259     }
260
261     /**
262     * @see
263     HttpSessionBindingListener#valueUnbound(HttpSessionBindingEvent)
264     */
265     public void valueUnbound(HttpSessionBindingEvent event) {
266         // TODO Auto-generated method stub
267     }
268 }
```

ComputeAnalytics.java

```
1 package analytics;
2
3 import java.io.File;
18
19 public class ComputeAnalytics {
20
21     public static synchronized void
computeOrders(HttpServletRequest req) {
22         HttpSession s = req.getSession();
23         if (s.getAttribute("order") == null) return;
24         Order order = (Order) s.getAttribute("order");
25         order.computeAllCosts();
26     }
27
28     public static synchronized void
computeUsersAnalytics(HttpServletRequest req) {
29         if (req.getAttribute("users") == null) return;
30
31         HttpSession s = req.getSession();
32         Account user = (Account) s.getAttribute("AUTH");
33         Map<String, long[]> accounts = (HashMap<String, long[]>)
req.getAttribute("users");
34         long[] avgTimes = new long[4];
35
36         if (req.getAttribute("allCheckoutAvgTime") == null) {
37             req.setAttribute("allCheckoutAvgTime", 0);
38         }
39
40         if (req.getAttribute("allAddToCartAvgTime") == null) {
41             req.setAttribute("allAddToCartAvgTime", 0);
42         }
43
44         if (Utils.isValidUser(user) &&
45             !accounts.containsKey(s.getId()) &&
46             s.getAttribute("checkoutAvgTime") != null &&
47             s.getAttribute("addToCartAvgTime") != null)
48         {
49             avgTimes[0] = (long)
s.getAttribute("checkoutAvgTime");
50             avgTimes[1] = (long)
s.getAttribute("addToCartAvgTime");
51             accounts.put(s.getId(), avgTimes);
52         }
53
```

ComputeAnalytics.java

```
54         int count = 1;
55         for (long[] accountAvgTimes: accounts.values()) {
56             long allCheckoutAvgTime = accountAvgTimes[0];
57             long allAddToCartAvgTime = accountAvgTimes[1];
58
59             req.setAttribute("allCheckoutAvgTime",
60 allCheckoutAvgTime/count);
61             req.setAttribute("allAddToCartAvgTime",
62 allAddToCartAvgTime/count);
63         }
64     }
65
66     public static synchronized void
67     addUserOrders(HttpServletRequest req) {
68         HttpSession currentSession = req.getSession();
69         Account ac = (Account)
70 currentSession.getAttribute("AUTH");
71         List<String> orders = (ArrayList<String>)
72 currentSession.getAttribute("orders");
73         String poDir = (String) req.getAttribute("poDir");
74
75         if (orders != null) {
76             OrderDAO orderDao = new OrderDAO(new File(poDir));
77 // req.setAttribute("orderDao", new OrderDAO(new
78 File(poDir)));
79             req.setAttribute("totalAllOrders",
80 orderDao.getP0s().length);
81
82             if (ac != null && Utils.isValidUser(ac)) {
83                 currentSession.setAttribute("totalOrders",
84 orderDao.getP0s(ac.getUsername()).length);
85
86                 for (File f: orderDao.getP0s(ac.getUsername())) {
87                     if (!orders.contains(f.getAbsolutePath())) {
88                         orders.add(f.getAbsolutePath());
89                     }
90                 }
91             }
92         }
93     }
94 }
```

ComputeAnalytics.java

```
90     public static synchronized void
computeAvgStartCheckoutTime(HttpServletRequest req) {
91         HttpSession currentSession = req.getSession();
92         String url = req.getRequestURI();
93         String path = url.replaceAll("/eFoods/", "");
94
95         long checkoutAvgTime = 0;
96         int checkoutAvgCount = 0;
97         Order order = (Order)
currentSession.getAttribute("order");
98
99         Map<String, long[]> accounts = (HashMap<String, long[]>)
req.getAttribute("users");
100
101         if (path.toLowerCase().contains("checkout") &&
102             req.getParameter("confirm") != null &&
103             !order.getItems().isEmpty() &&
104             currentSession.getAttribute("checkoutTime") ==
null)
105         {
106             currentSession.setAttribute("checkoutTime", new
Date());
107
108             if (currentSession.getAttribute("checkoutAvgTime") !=
null) {
109                 checkoutAvgTime = (long)
currentSession.getAttribute("checkoutAvgTime");
110             }
111
112             if (currentSession.getAttribute("checkoutAvgCount") !=
null) {
113                 checkoutAvgCount = (int)
currentSession.getAttribute("checkoutAvgCount");
114             }
115
116             Date startTime = (Date)
currentSession.getAttribute("startTime");
117             Date checkoutTime = (Date)
currentSession.getAttribute("checkoutTime");
118
119             long diff = checkoutTime.getTime() -
startTime.getTime();
120             long seconds = TimeUnit.MILLISECONDS.toSeconds(diff);
121
```

ComputeAnalytics.java

```
122         checkoutAvgCount++;
123         checkoutAvgTime = (checkoutAvgTime + seconds) /
checkoutAvgCount;
124
125         currentSession.setAttribute("checkoutAvgTime",
checkoutAvgTime);
126         currentSession.setAttribute("checkoutAvgCount",
checkoutAvgCount);
127
128         accounts.put(currentSession.getId(), new long[]{(long)
currentSession.getAttribute("checkoutAvgTime")});
129     }
130 }
131
132 public static synchronized void
computeAvgAddToCartTime(HttpServletRequest req) {
133     HttpSession currentSession = req.getSession();
134     String url = req.getRequestURI();
135     String path = url.replaceAll("/eFoods/", "");
136
137     long addToCartAvgTime = 0;
138     int addToCartAvgCount = 0;
139     Order order = (Order)
currentSession.getAttribute("order");
140     Map<String, long[]> accounts = (HashMap<String, long[]>)
req.getAttribute("users");
141
142     if (path.toLowerCase().contains("api/cart") &&
143         req.getParameter("addToCart") != null &&
144         order.getItems().isEmpty() &&
145         currentSession.getAttribute("addToCartTime") ==
null)
146     {
147         currentSession.setAttribute("addToCartTime", new
Date());
148
149         if (currentSession.getAttribute("addToCartAvgTime") !=
null) {
150             addToCartAvgTime = (long)
currentSession.getAttribute("addToCartAvgTime");
151         }
152
153         if (currentSession.getAttribute("addToCartAvgCount") !=
null) {
```


ComputeAnalytics.java

```
154         addToCartAvgCount = (int)
currentSession.getAttribute("addToCartAvgCount");
155     }
156
157     Date startTime = (Date)
currentSession.getAttribute("startTime");
158     Date addToCartTime = (Date)
currentSession.getAttribute("addToCartTime");
159
160     long diff = addToCartTime.getTime() -
startTime.getTime();
161     long seconds = TimeUnit.MILLISECONDS.toSeconds(diff);
162
163     addToCartAvgCount++;
164     addToCartAvgTime = (addToCartAvgTime + seconds) /
addToCartAvgCount;
165
166     currentSession.setAttribute("addToCartAvgTime",
addToCartAvgTime);
167     currentSession.setAttribute("addToCartAvgCount",
addToCartAvgCount);
168
169     accounts.put(currentSession.getId(), new long[]{(long)
currentSession.getAttribute("addToCartAvgTime")});
170     }
171 }
172 }
173
```



Adhoc

AuthFilter.java

```
1 package adhoc;
2
3 import java.io.IOException;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22 /**
23  * Servlet Filter implementation class AuthFilter
24  */
25 @WebFilter("/AuthFilter")
26 public class AuthFilter implements Filter {
27
28     /**
29      * Default constructor.
30      */
31     public AuthFilter() {
32         // TODO Auto-generated constructor stub
33     }
34
35     /**
36      * @see Filter#destroy()
37      */
38     public void destroy() {
39         // TODO Auto-generated method stub
40     }
41
42     /**
43      * @see Filter#doFilter(ServletRequest, ServletResponse,
44      * FilterChain)
45      */
46     public void doFilter(ServletRequest request, ServletResponse
47     response, FilterChain chain) throws IOException, ServletException
48     {
49         // TODO Auto-generated method stub
50         HttpServletRequest req = (HttpServletRequest) request;
51         HttpServletResponse resp = (HttpServletResponse) response;
52         HttpSession s = req.getSession();
53         String url = req.getRequestURI();
54         Account user = (Account) s.getAttribute("AUTH");
55
56         String path = url.replaceAll("/eFoods/", "");
57
58         String userName = request.getParameter("name");
59         String username = request.getParameter("user");
60
61         if (!Utils.isValidUser(user)) {
```

AuthFilter.java

```
59         if (userName != null && username != null) {
60             user.setName(userName);
61             user.setUsername(username);
62         }
63     }
64
65     if (path.contains("Checkout") || path.contains("Login") ||
66         path.contains("PurchaseOrders") ||
67         req.getParameter("doLogin") != null)
68     {
69         if (!Utils.isValidUser(user)) {
70             if (userName != null && username != null) {
71                 user.setName(userName);
72                 user.setUsername(username);
73             }
74
75             String redirectUrl = "https://www.eecs.yorku.ca/
~roumani/servers/auth/oauth.cgi";
76             String params = "?back=" + req.getRequestURL();
77             resp.sendRedirect(redirectUrl + params);
78             return;
79         }
80     }
81
82     showAds(req);
83
84     chain.doFilter(request, response);
85 }
86
87 /**
88  * @see Filter#init(FilterConfig)
89  */
90 public void init(FilterConfig fConfig) throws ServletException
91 {
92     // TODO Auto-generated method stub
93 }
94
95 private static synchronized void showAds(HttpServletRequest
req) {
96     String addToCart = req.getParameter("addToCart");
97     HttpSession s = req.getSession();
98
99     if (addToCart != null) {
100         String itemToSellAds = "2002H712";
```

AuthFilter.java

```
100         String onItemShowAds = "1409S413";
101         if (addToCart.equals(onItemShowAds)) {
102             try {
103                 Item adItem =
104                     Model.getInstance().getFood(itemToSellAds);
105                 if (s.getAttribute("adItem") != null) {
106                     s.setAttribute("adItem", adItem);
107                 }
108             } catch (Exception e) {
109                 // TODO Auto-generated catch block
110                 e.printStackTrace();
111             }
112         }
113     }
114 }
115 }
116
```



View

web.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
  java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
3   <display-name>eFoods</display-name>
4   <welcome-file-list>
5     <welcome-file>Home</welcome-file>
6   </welcome-file-list>
7   <listener>
8     <listener-class>analytics.Monitor</listener-class>
9   </listener>
10  <error-page>
11    <error-code>404</error-code>
12    <location>/WEB-INF/pages/Error404.jsp</location>
13  </error-page>
14  <filter>
15    <filter-name>AuthFilter</filter-name>
16    <filter-class>adhoc.AuthFilter</filter-class>
17  </filter>
18  <filter-mapping>
19    <filter-name>AuthFilter</filter-name>
20    <url-pattern>*</url-pattern>
21  </filter-mapping>
22 </web-app>
```



Includes

Header.jspx

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
3   xmlns:c="http://java.sun.com/jsp/jstl/core"
4   xmlns:fmt="http://java.sun.com/jsp/jstl/fmt"
5   version="2.0">
6
7   <head>
8     <meta charset="UTF-8" />
9     <meta name="viewport" content="width=device-width, initial-
10    scale=1" />
11     <title>${param.title}</title>
12     <!-- CSS -->
13     <link rel="stylesheet" href="${param.rootURLPath}/res/css/
14    main.css" />
15     <link rel="stylesheet" href="${param.rootURLPath}/res/css/
16    bulma.min.css" />
17   </head>
18 </jsp:root>
```

Nav.jspx

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
3   xmlns:c="http://java.sun.com/jsp/jstl/core"
4   xmlns:fmt="http://java.sun.com/jsp/jstl/fmt" version="2.0">
5
6   <nav class="navbar has-shadow is-spaced"
7     style="background: antiquewhite; margin-bottom: 10px;"
8     role="navigation" aria-label="main navigation">
9     <div class="container">
10       <div class="navbar-brand">
11         <a class="navbar-item" href="{param.rootURLPath}/
12         Home"> 
15         </a> <a role="button" class="navbar-burger burger"
16           aria-label="menu"
17           aria-expanded="false" data-target="navMenu">
18         <span
19           aria-hidden="true"></span> <span aria-
20             hidden="true"></span> <span
21               aria-hidden="true"></span>
22             </a>
23           </div>
24
25           <div id="navMenu" class="navbar-menu">
26             <!-- Left Nav items -->
27             <div class="navbar-start">
28               <a class="navbar-item is-boxed" href="{
29                 param.rootURLPath}/Home">Catalog</a>
30               <a class="navbar-item is-boxed" href="{
31                 param.rootURLPath}/PurchaseOrders">Purchase Orders</a>
32             </div>
33
34             <!-- Right Nav items -->
35             <div class="navbar-end">
36               <div class="navbar-item">
37                 <div class="field is-horizontal is-hidden-
38                   mobile" style="margin-right: 10px; margin-bottom: 0px;">
39                   <input type="text" class="input"
40                     placeholder="Search" id="sQuery" style="margin-right: 10px;" />
41                   <input type="submit" class="button is-
42                     primary" value="Search" id="doSearch" onclick="gotoSearchPg()" />
43                 </div>
44               <div class="buttons">
```

Nav.jsp

```
35         <a class="button is-primary" href="$  
    {param.rootURLPath}/Cart"> <strong>Cart</strong></a>  
36         <c:if test="${!empty  
    order.getItems()}">  
37             <a class="button is-info" href="$  
    {param.rootURLPath}/Checkout"> <strong>Checkout</strong></a>  
38         </c:if>  
39         <c:choose>  
40             <c:when test="${!empty AUTH.name}">  
41                 <a class="button is-link"  
    href="${param.rootURLPath}/PurchaseOrders">Hello ${AUTH.name}</a>  
42             </c:when>  
43             <c:otherwise>  
44                 <a class="button is-light"  
    onclick="doLogin()">Login</a>  
45             </c:otherwise>  
46             </c:choose>  
47         </div>  
48     </div>  
49 </div>  
50 </div>  
51 </div>  
52 </nav>  
53  
54 </jsp:root>  
55
```

Footer.jspx

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
3   xmlns:c="http://java.sun.com/jsp/jstl/core"
4   xmlns:fmt="http://java.sun.com/jsp/jstl/fmt" version="2.0">
5
6
7   <!-- SHow Ads -->
8   <c:if test="${!empty adItem}">
9     <section class="container">
10       <c:set var="item" value="${adItem}" />
11       <jsp:include page="Item.jspx">
12         <jsp:include page="Item.jspx">
13           <jsp:param name="name" value="${item.name}" />
14           <jsp:param name="number" value="$
15 {item.number}" />
16           <jsp:param name="price" value="${item.price}" /
17 >
18           <jsp:param name="unit" value="$
19 {item.quantity}"/>
20           <jsp:param name="catId" value="${item.catID}"/>
21         </jsp:include>
22       </jsp:include>
23     </section>
24   </c:if>
25
26   <!-- JS -->
27   <script type="text/javascript" src="${param.rootURLPath}/res/
28 js/main.js"><!-- --></script>
29
30   <footer class="footer">
31     <div class="content has-text-centered">
32       <p>
33         <strong>eFoods</strong> by Haseeb Ahmad, Talha
34         Mahmood.
35       </p>
36       <p>
37         <a href="https://bulma.io"> 
42         </a>
43       </p>
44     </div>
```

Footer.jspx

```
38     </footer>
39
40 </jsp:root>
41
```

Error.jspx

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
3   xmlns:c="http://java.sun.com/jsp/jstl/core"
4   xmlns:fmt="http://java.sun.com/jsp/jstl/fmt"
5   version="2.0">
6
7   <c:choose>
8     <c:when test="{not empty error}">
9       <div class="alert alert-danger error" role="alert">${
{error}}</div>
10     </c:when>
11     <c:otherwise>
12       <div class="error hidden" role="alert"><!-- --></div>
13     </c:otherwise>
14   </c:choose>
15
16 </jsp:root>
17
```

Item.java

```
1 package model.catalog;
2
3 import javax.xml.bind.annotation.XmlAttribute;
4
5 @XmlRootElement(name="item")
6 public class Item {
7
8     private String name, number, unit;
9
10    private double price, totalPrice;
11    private int quantity, catID;
12
13    public Item() {
14    }
15
16    public Item(String name, double price, int quantity, String
17    number, String unit) throws Exception {
18        this(name, price, quantity, number, 0, unit);
19    }
20
21    public Item(String name, double price, int quantity, String
22    number, int catId, String unit) throws Exception {
23        setName(name);
24        setPrice(price);
25        setQuantity(quantity);
26        setNumber(number);
27        setCatID(catID);
28        setUnit(unit);
29    }
30
31    @XmlElement(name="name")
32    public String getName() {
33        return name;
34    }
35
36    public void setName(String name) {
37        this.name = name;
38    }
39
40    @XmlElement(name="price")
41    public double getPrice() {
42        return Utils.round(price, 2);
43    }
44
45 }
46
```

Item.java

```
47     public void setPrice(double price) throws Exception {
48         if (price < 0.0) {
49             throw new IllegalArgumentException("Price < 0.0");
50         }
51         this.price = Utils.round(price, 2);
52     }
53
54     @XmlElement(name="quantity")
55     public int getQuantity() {
56         return quantity;
57     }
58
59     public void setQuantity(int quantity) throws Exception {
60         if (quantity < 0) {
61             throw new IllegalArgumentException("Qty < 0");
62         }
63         this.quantity = quantity;
64     }
65
66     public void setQuantity(String quantity) throws Exception {
67         int qty = Integer.parseInt(quantity);
68         this.setQuantity(qty);
69     }
70
71     @XmlAttribute(name="number")
72     public String getNumber() {
73         return number;
74     }
75
76     public void setNumber(String number) {
77         this.number = number;
78     }
79
80     public int getCatID() {
81         return catID;
82     }
83
84     public void setCatID(int catID) {
85         this.catID = catID;
86     }
87
88     public String getUnit() {
89         return unit;
90     }
```


Item.java

```
91
92     public void setUnit(String unit) {
93         this.unit = unit;
94     }
95
96     public void setTotalPrice(double totalPrice) {
97         this.totalPrice = Utils.round(totalPrice, 2);
98     }
99
100     @XmlElement(name="extended")
101     public double getTotalPrice() {
102         return Utils.round(totalPrice, 2);
103     }
104
105     public void increaseQty() {
106         this.quantity++;
107     }
108
109     public void increaseQty(int qty) {
110         this.quantity += qty;
111     }
112
113     public void increaseQty(String qtyStr) throws Exception {
114         int qty = Integer.parseInt(qtyStr);
115         this.setQuantity(qty);
116     }
117
118     public double computeTotalPrice() {
119         this.setTotalPrice(this.price * this.quantity);
120         return Utils.round(this.totalPrice, 2);
121     }
122
123     public String toString() {
124         return String.format("Name: %s | Price: %s | Quantity: %s
| Number: %s | CatId: %s | Unit: %s \n", name, price, quantity,
number, catID, unit);
125     }
126
127     @Override
128     public boolean equals(Object obj) {
129         if (this == obj) return true;
130         if (obj == null) return false;
131         if (!(obj instanceof Item)) return false;
132
```

Item.java

```
133     Item other = (Item) obj;
134     if (!this.number.equals(other.number)) {
135         return false;
136     }
137
138     return true;
139 }
140
141
142 }
143
```

CatItemsCard.jspx

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
3   xmlns:c="http://java.sun.com/jsp/jstl/core"
4   xmlns:fn="http://java.sun.com/jsp/jstl/functions"
5   xmlns:fmt="http://java.sun.com/jsp/jstl/fmt"
6   version="2.0">
7
8   <div class="card">
9     <header class="card-header">
10       <c:choose>
11         <c:when test="${param.search == 1}">
12           <p class="card-header-title">Search Results</p>
13         </c:when>
14         <c:otherwise>
15           <p class="card-header-title">${param.catName}</p>
16         </c:choose>
17         <a class="card-header-icon secondary"
18           style="border-style: solid; border-width: 1px;" aria-label="more
19           options" id="view-cat-item-${param.catId}">
20           <strong>View All</strong>
21         </a>
22       </c:otherwise>
23     </c:choose>
24   </header>
25
26   <div class="card-content">
27     <div class="content">
28       <div class="columns is-multiline">
29         <c:forEach var="item" items="${items}"
30           varStatus="loop">
31           <c:choose>
32             <c:when test="${param.limit gt 0}">
33               <c:if test="${loop.index le
34                 param.limit}">
35                 <div class="column is-3">
36                   <jsp:include
37                     page="Item.jspx">
38                     <jsp:param name="name"
39                       value="${item.name}" />
40                     <jsp:param
41                       name="number" value="${item.number}" />
42                     <jsp:param name="price"
43                       value="${item.price}" />
44                     <jsp:param name="unit"
```

CatItemsCard.jspx

```

    value="${item.quantity}"/>
36                                     <jsp:param name="catId"
    value="${item.catID}"/>
37                                     <jsp:param name="unit"
    value="${item.unit}"/>
38                                     <jsp:param name="qty"
    value="${item.quantity}"/>
39                                     </jsp:include>
40                                     </div>
41                                     </c:if>
42                                 </c:when>
43                                 <c:otherwise>
44                                     <div class="column is-3">
45                                         <jsp:include page="Item.jspx">
46                                             <jsp:param name="name"
    value="${item.name}" />
47                                             <jsp:param
    name="number" value="${item.number}" />
48                                             <jsp:param name="price"
    value="${item.price}" />
49                                             <jsp:param name="unit"
    value="${item.quantity}"/>
50                                             <jsp:param name="catId"
    value="${item.catID}"/>
51                                             <jsp:param name="unit"
    value="${item.unit}"/>
52                                             <jsp:param name="qty"
    value="${item.quantity}"/>
53                                         </jsp:include>
54                                         </div>
55                                     </c:otherwise>
56                                 </c:choose>
57                             </c:forEach>
58                         </div>
59                     </div>
60                 </div>
61
62                 <!-- <footer class="card-footer has-text-centered">
63                     <a href="#" class="card-footer-item has-text-
centered">View All</a>
64                 </footer> -->
65             </div>
66
67 </jsp:root>

```

CatItemsCard.jspx



Pages

Home.jspx

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
3   xmlns:c="http://java.sun.com/jsp/jstl/core"
4   xmlns:fn="http://java.sun.com/jsp/jstl/functions"
5   xmlns:fmt="http://java.sun.com/jsp/jstl/fmt"
6   version="2.0">
7   <jsp:directive.page contentType="text/html; charset=UTF-8"
8     pageEncoding="UTF-8" session="false"/>
9   <jsp:output doctype-root-element="html"
10     doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
11     doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd"
12     omit-xml-declaration="true" />
13 <![CDATA[<!DOCTYPE html>]]>
14 <html xmlns="http://www.w3.org/1999/xhtml">
15   <c:set var="rootURLPath" value="$
{pageContext.request.contextPath}"></c:set>
16
17   <jsp:include page="../includes/Header.jspx">
18     <jsp:param name="title" value="Foods R Us!" />
19     <jsp:param name="rootURLPath" value="${rootURLPath}" />
20   </jsp:include>
21
22   <body>
23     <jsp:include page="../includes/Nav.jspx">
24       <jsp:param name="rootURLPath" value="${rootURLPath}" /
25     </jsp:include>
26     <main class="bd-main">
27       <section class="hero is-medium is-primary is-bold"
28         style="background-image: url(res/img/stand.png);
29         background-position: center;
30         background-size: contain;
31         background-repeat: no-repeat;">
32         <div class="hero-body">
33           <div class="container">
34             <h1 class="title has-text-light">Food R Us!</
h1>
35             <h2 class="subtitle has-text-light">An
eCommerce Site</h2>
36           </div>
37         </div>
38       </section>
39     <div class="container">
```

Home.jspx

[illegible]

Home.jspx

```

71         <span>${cat.name}</span>
72     </a>
73 </li>
74 </c:forEach>
75 </ul>
76 </c:if>
77 </div>
78
79 <c:if test="${!empty catsWithFoods}">
80     <div class="tab-content">
81         <c:forEach items="${catsWithFoods}"
var="cat" varStatus="loop">
82             <div class="tab-pane level"
id="pane-${cat.key.id}">
83                 <c:set var="items" value="$
{cat.value}" scope="request" />
84                 <jsp:include page="../../
includes/CatItemsCard.jspx">
85                     <jsp:param name="catName"
value="${cat.key.name}" />
86                     <jsp:param name="catId"
value="${cat.key.id}" />
87                     <jsp:param name="items"
value="${cat.value}" />
88                     <jsp:param name="search"
value="0" />
89                 </jsp:include>
90             </div>
91         </c:forEach>
92     </div>
93 </c:if>
94 </section>
95
96 <section id="page-products">
97     <c:if test="${!empty catsWithFoods}">
98         <c:forEach items="${catsWithFoods}"
var="cat" varStatus="loop">
99             <c:set var="items" value="$
{cat.value}" scope="request" />
100             <jsp:include page="../../includes/
CatItemsCard.jspx">
101                 <jsp:param name="catName" value="$
{cat.key.name}" />
102                 <jsp:param name="catId" value="$

```

Home.jspx

```
103 {cat.key.id}" />
104     <jsp:param name="limit"
value="3" />
105     <jsp:param name="search"
value="0" />
106     <jsp:param name="items" value="$
{cat.value}" />
107     </jsp:include>
108     <br />
109     </c:forEach>
110 </c:if>
111 </section>
112 </div>
113 </main>
114 <jsp:include page="../../includes/Footer.jspx">
115     <jsp:param name="rootURLPath" value="${rootURLPath}" /
>
116 </jsp:include>
117 </body>
118 </html>
119 </jsp:root>
```

Admin.jspx

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
3   xmlns:c="http://java.sun.com/jsp/jstl/core"
4   xmlns:fn="http://java.sun.com/jsp/jstl/functions"
5   xmlns:fmt="http://java.sun.com/jsp/jstl/fmt"
6   version="2.0">
7   <jsp:directive.page contentType="text/html; charset=UTF-8"
8     pageEncoding="UTF-8" session="false"/>
9   <jsp:output doctype-root-element="html"
10     doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
11     doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
12       transitional.dtd"
13     omit-xml-declaration="true" />
14 <![CDATA[<!DOCTYPE html>]]>
15 <html xmlns="http://www.w3.org/1999/xhtml">
16   <c:set var="rootURLPath" value="$
17     {pageContext.request.contextPath}"></c:set>
18   <jsp:include page="../../includes/Header.jspx">
19     <jsp:param name="title" value="Foods R Us!" />
20     <jsp:param name="rootURLPath" value="${rootURLPath}" />
21   </jsp:include>
22   <body>
23     <jsp:include page="../../includes/Nav.jspx">
24       <jsp:param name="rootURLPath" value="${rootURLPath}" />
25     </jsp:include>
26     <main class="bd-main">
27       <section class="hero is-small is-primary is-bold">
28         <div class="hero-body">
29           <div class="container">
30             <h1 class="title has-text-light">Admin!</h1>
31           </div>
32         </div>
33       </section>
34       <div class="container">
35         <section>
36           <c:set var="error" value="${error}"
37             scope="request" />
38           <jsp:include page="../../includes/Error.jspx" />
39         </section>
40         <section id="current-user-analytics" style="margin-
```

Admin.jspx

```
bottom: 20px">
42         <h1 class="is-size-2">Current User Analytics</
h1>
43         <table
44             class="table is-bordered is-striped is-
hoverable is-fullwidth"
45             id="cartItems">
46             <thead>
47                 <tr>
48                     <th>Name</th>
49                     <th>Time (Seconds)</th>
50                 </tr>
51             </thead>
52             <tbody>
53                 <tr>
54                     <td>Add to cart time</td>
55                     <td>${addToCartAvgTime}</td>
56                 </tr>
57
58                 <tr>
59                     <td>Checkout time</td>
60                     <td>${checkoutAvgTime}</td>
61                 </tr>
62             </tbody>
63         </table>
64     </section>
65
66     <section id="all-users-analytics">
67         <h1 class="is-size-2">All Users Analytics</h1>
68         <table
69             class="table is-bordered is-striped is-
hoverable is-fullwidth"
70             id="cartItems">
71             <thead>
72                 <tr>
73                     <th>Average add to cart time
(Seconds)</th>
74                     <th>Average checkout time
(Seconds)</th>
75                 </tr>
76             </thead>
77             <tbody>
78                 <tr>
79                     <td>${allCheckoutAvgTime}</td>
```

Admin.jspx

```
80         <td>${allAddToCartAvgTime}</td>
81     </tr>
82 </tbody>
83 </table>
84 </section>
85 </div>
86 </main>
87 <jsp:include page="../../includes/Footer.jspx">
88     <jsp:param name="rootURLPath" value="${rootURLPath}" />
89 </jsp:include>
90 </body>
91 </html>
92 </jsp:root>
93
```

Search.jspx

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
3   xmlns:c="http://java.sun.com/jsp/jstl/core"
4   xmlns:fn="http://java.sun.com/jsp/jstl/functions"
5   xmlns:fmt="http://java.sun.com/jsp/jstl/fmt"
6   version="2.0">
7   <jsp:directive.page contentType="text/html; charset=UTF-8"
8     pageEncoding="UTF-8" session="false"/>
9   <jsp:output doctype-root-element="html"
10     doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
11     doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
    transitional.dtd"
12     omit-xml-declaration="true" />
13 <![CDATA[<!DOCTYPE html>]]>
14 <html xmlns="http://www.w3.org/1999/xhtml">
15   <c:set var="rootURLPath" value="$
    {pageContext.request.contextPath}"></c:set>
16
17   <jsp:include page="../../includes/Header.jspx">
18     <jsp:param name="title" value="Foods R Us!" />
19     <jsp:param name="rootURLPath" value="${rootURLPath}" />
20   </jsp:include>
21
22   <body>
23     <jsp:include page="../../includes/Nav.jspx">
24       <jsp:param name="rootURLPath" value="${rootURLPath}" />
25     </jsp:include>
26     <main class="bd-main">
27       <section class="hero is-primary is-bold">
28         <div class="hero-body">
29           <div class="container">
30             <h1 class="title">Search</h1>
31           </div>
32         </div>
33       </section>
34       <div class="container">
35         <section>
36           <c:set var="error" value="${error}"
    scope="request" />
37           <jsp:include page="../../includes/Error.jspx" />
38         </section>
39         <section>
40           <form action="Search" style="margin-bottom:
    15px;">
```

Search.jspx

```
41         <div class="field has-addons">
42             <div class="control" style="width:
100%;">
43                 <input class="input" type="text"
name="query" placeholder="Find an Item" value="${query}" />
44             </div>
45             <div class="control">
46                 <input type="submit"
name="doSearch" class="button is-info" value="Search" />
47             </div>
48         </div>
49     </form>
50 </section>
51
52     <section id="page-products">
53         <c:if test="${!empty searchItems}">
54             <c:set var="items" value="${searchItems}"
scope="request" />
55             <jsp:include page="../includes/
CatItemsCard.jspx">
56                 <jsp:param name="catName" value="" />
57                 <jsp:param name="catId" value="" />
58                 <jsp:param name="search" value="1" />
59                 <jsp:param name="items" value="$
{searchItems}" />
60             </jsp:include>
61             <br />
62         </c:if>
63     </section>
64 </div>
65 </main>
66 <jsp:include page="../includes/Footer.jspx">
67     <jsp:param name="rootURLPath" value="${rootURLPath}" />
68 </jsp:include>
69 </body>
70 </html>
71 </jsp:root>
72
```

Cart.jspx

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
3   xmlns:c="http://java.sun.com/jsp/jstl/core"
4   xmlns:fn="http://java.sun.com/jsp/jstl/functions"
5   xmlns:fmt="http://java.sun.com/jsp/jstl/fmt" version="2.0">
6   <jsp:directive.page contentType="text/html; charset=UTF-8"
7     pageEncoding="UTF-8" session="false" />
8   <jsp:output doctype-root-element="html"
9     doctype-public "-//W3C//DTD XHTML 1.0 Transitional//EN"
10    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
    transitional.dtd"
11    omit-xml-declaration="true" />
12   <![CDATA[<!DOCTYPE html>]]>
13   <html xmlns="http://www.w3.org/1999/xhtml">
14 <c:set var="rootURLPath" value="$
    {pageContext.request.contextPath}"></c:set>
15
16 <jsp:include page="../includes/Header.jspx">
17   <jsp:param name="title" value="Foods R Us!" />
18   <jsp:param name="rootURLPath" value="${rootURLPath}" />
19 </jsp:include>
20
21 <body>
22   <jsp:include page="../includes/Nav.jspx">
23     <jsp:param name="rootURLPath" value="${rootURLPath}" />
24   </jsp:include>
25
26   <fmt:setLocale value = "en_US"/>
27
28   <main class="bd-main">
29   <section class="hero is-small is-primary is-bold">
30     <div class="hero-body">
31       <div class="container">
32         <h1 class="title">Shopping Cart</h1>
33       </div>
34     </div>
35   </section>
36   <div class="container">
37     <section>
38       <c:set var="error" value="${error}" scope="request" />
39       <jsp:include page="../includes/Error.jspx" />
40     </section>
41     <section id="cart-products">
42       <c:if test="${!empty order.getItems()}">
```


Cart.jspx

```
43         <table
44             class="table is-bordered is-striped is-
45             hoverable is-fullwidth"
46             id="cartItems">
47             <thead>
48                 <tr>
49                     <th>Product Number</th>
50                     <th>Product Name</th>
51                     <th>Price</th>
52                     <th>Extended Price</th>
53                     <th>Quantity</th>
54                     <th>Actions</th>
55                 </tr>
56             </thead>
57             <tbody>
58                 <c:forEach items="${order.getItems()}"
59                     var="product"
60                     varStatus="loop">
61                     <tr id="p-${product.number}">
62                         <td>${product.number}</td>
63                         <td>${product.name}</td>
64                         <td><fmt:formatNumber
65                             type="currency" value="${product.price}"></fmt:formatNumber></td>
66                         <td>${product.totalPrice}</td>
67                         <td><input type="text"
68                             class="input" placeholder=""
69                             id="qty-${product.number}"
70                             value="${product.quantity}" /></td>
71                         <td>
72                             <button style="margin-right:
73                                 10px;" class="button is-primary"
74                                 onclick="updateQty(this)">Update</button>
75
76                             <button style="" class="button
77                                 is-danger" onclick="deleteCartItem(this)">Delete</button>
78                         </td>
79                     </tr>
80                 </c:forEach>
81             </tbody>
82         </table>
83
84         <div class="has-text-right has-background-light"
85             style="margin-bottom: 15px; padding: 10px;">
86             <p class="is-size-3 is-capitalized has-text-
```

Cart.jspx

```
weight-bold">Summary</p>
78         <p class="is-size-4">
79             <strong>Total: </strong>
80             <span><fmt:formatNumber type="currency"
value="\${order.total}"></fmt:formatNumber></span>
81         </p>
82         <p class="is-size-4">
83             <strong>Shipping: </strong>
84             <span><fmt:formatNumber type="currency"
value="\${order.shipping}"></fmt:formatNumber></span>
85         </p>
86         <p class="is-size-4">
87             <strong>HST (incl. Shipping): </strong>
88             <span><fmt:formatNumber type="currency"
value="\${order.HST}"></fmt:formatNumber></span>
89         </p>
90         <p class="is-size-4 is-capitalized">
91             <strong>Grand Total: </strong>
92             <span><fmt:formatNumber type="currency"
value="\${order.grandTotal}"></fmt:formatNumber></span>
93         </p>
94     </div>
95
96     <div class="field is-grouped" style="justify-
content: flex-end; margin-bottom: 10px;">
97         <div class="control">
98             <button class="button is-primary"
onclick="updateItemsQty()">Update</button>
99         </div>
100        <div class="control">
101            <button class="button is-link"
onclick="gotoHomePage()">Continue Shopping</button>
102        </div>
103
104        <div class="control">
105            <button class="button is-info"
onclick="doCheckout()">Checkout</button>
106        </div>
107    </div>
108    </c:if>
109
110    <c:if test="\${empty order.getItems()}">
111        <p class="is-size-3 has-text-centered is-
capitalized has-text-weight-bold">Empty Cart</p>
```

Cart.jspx

```
112         </c:if>
113     </section>
114 </div>
115 </main>
116
117 <jsp:include page="../../includes/Footer.jspx">
118     <jsp:param name="rootURLPath" value="${rootURLPath}" />
119 </jsp:include>
120 </body>
121 </html>
122 </jsp:root>
123
```

Checkout.jspx

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
3   xmlns:c="http://java.sun.com/jsp/jstl/core"
4   xmlns:fn="http://java.sun.com/jsp/jstl/functions"
5   xmlns:fmt="http://java.sun.com/jsp/jstl/fmt" version="2.0">
6   <jsp:directive.page contentType="text/html; charset=UTF-8"
7     pageEncoding="UTF-8" session="false" />
8   <jsp:output doctype-root-element="html"
9     doctype-public "-//W3C//DTD XHTML 1.0 Transitional//EN"
10    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
    transitional.dtd"
11    omit-xml-declaration="true" />
12   <![CDATA[<!DOCTYPE html>]]>
13   <html xmlns="http://www.w3.org/1999/xhtml">
14 <c:set var="rootURLPath" value="$
    {pageContext.request.contextPath}"></c:set>
15
16 <jsp:include page="../includes/Header.jspx">
17   <jsp:param name="title" value="Foods R Us!" />
18   <jsp:param name="rootURLPath" value="{rootURLPath}" />
19 </jsp:include>
20
21 <body>
22   <jsp:include page="../includes/Nav.jspx">
23     <jsp:param name="rootURLPath" value="{rootURLPath}" />
24   </jsp:include>
25
26   <fmt:setLocale value = "en_US"/>
27
28   <main class="bd-main">
29   <section class="hero is-small is-primary is-bold">
30     <div class="hero-body">
31       <div class="container">
32         <h1 class="title">Purchase Order Confirmation</h1>
33       </div>
34     </div>
35   </section>
36   <div class="container">
37     <section>
38       <c:set var="error" value="{error}" scope="request" />
39       <jsp:include page="../includes/Error.jspx" />
40     </section>
41     <c:if test="{!empty orderCreated}">
42       <p class="is-size-1">The Purchase Order has been
```

Checkout.jspx

```
completed!</p>
43     <p class="is-size-2">Thank you!</p>
44     <p class="is-size-4">Here is the <a class="is-link"
href="${rootURLPath}/PurchaseOrders?orderNum=${
{orderCreated}}">link</a> to view this purchase order.</p>
45     </c:if>
46     <c:if test="${empty orderCreated}">
47         <section id="cart-products">
48             <c:if test="${!empty order.getItems()}">
49                 <table
50                     class="table is-bordered is-striped is-
hoverable is-fullwidth"
51                     id="cartItems">
52                     <thead>
53                         <tr>
54                             <th>Product Number</th>
55                             <th>Product Name</th>
56                             <th>Price</th>
57                             <th>Extended Price</th>
58                             <th>Quantity</th>
59                         </tr>
60                     </thead>
61                     <tbody>
62                         <c:forEach items="${order.getItems()}"
var="product"
63                             varStatus="loop">
64                             <tr id="p-${product.number}">
65                                 <td>${product.number}</td>
66                                 <td>${product.name}</td>
67                                 <td><fmt:formatNumber
type="currency" value="${product.price}"></fmt:formatNumber></td>
68                                 <td><fmt:formatNumber
type="currency" value="${product.totalPrice}"></
fmt:formatNumber></td>
69                                 <td><fmt:formatNumber
type="currency" value="${product.quantity}"></fmt:formatNumber></
td>
70                             </tr>
71                         </c:forEach>
72                     </tbody>
73                 </table>
74
75                 <div class="has-text-right has-background-
light" style="margin-bottom: 15px; padding: 10px;">
```

Checkout.jspx

```
76         <p class="is-size-3 is-capitalized has-  
text-weight-bold">Summary</p>  
77         <p class="is-size-4">  
78             <strong>Total: </strong>  
79             <span><fmt:formatNumber  
type="currency" value="{order.total}"></fmt:formatNumber></span>  
80         </p>  
81         <p class="is-size-4">  
82             <strong>Shipping: </strong>  
83             <span><fmt:formatNumber  
type="currency" value="{order.shipping}"></fmt:formatNumber></  
span>  
84         </p>  
85         <p class="is-size-4">  
86             <strong>HST (incl. Shipping): </  
strong>  
87             <span><fmt:formatNumber  
type="currency" value="{order.HST}"></fmt:formatNumber></span>  
88         </p>  
89         <p class="is-size-4 is-capitalized">  
90             <strong>Grand Total: </strong>  
91             <span><fmt:formatNumber  
type="currency" value="{order.grandTotal}"></fmt:formatNumber></  
span>  
92         </p>  
93     </div>  
94  
95     <form action="Checkout">  
96         <!-- <div class="field">  
97             <div class="control">  
98                 <label class="checkbox">  
99                     <input type="checkbox" />  
100                     I agree to the <a  
href="#">terms and conditions</a>  
101                         </label>  
102                     </div>  
103                 </div>  
104  
105                 <div class="field">  
106                     <div class="control">  
107                         <label class="radio">  
108                             <input type="radio" id="yes"  
name="confirm" />  
109                             Yes
```

Checkout.jspx

```
110         </label>
111
112         <label class="radio">
113             <input type="radio" id="no"
name="confirm" />
114             No
115         </label>
116     </div>
117 </div> -->
118
119     <div class="field is-grouped">
120         <div class="control">
121             <input type="submit"
value="Confirm" name="confirm" class="button is-link" />
122         </div>
123         <div class="control">
124             <input type="submit"
value="Cancel" name="cancel" class="button is-link" />
125         </div>
126     </div>
127 </form>
128 </c:if>
129
130     <c:if test="${empty order.getItems()}">
131         <p class="is-size-3 has-text-centered is-
capitalized has-text-weight-bold">Empty Cart</p>
132     </c:if>
133 </section>
134 </c:if>
135 </div>
136 </main>
137
138 <jsp:include page="../../includes/Footer.jspx">
139     <jsp:param name="rootURLPath" value="${rootURLPath}" />
140 </jsp:include>
141 </body>
142 </html>
143 </jsp:root>
144
```

PurchaseOrders.jspx

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
3   xmlns:c="http://java.sun.com/jsp/jstl/core"
4   xmlns:fn="http://java.sun.com/jsp/jstl/functions"
5   xmlns:fmt="http://java.sun.com/jsp/jstl/fmt" version="2.0">
6   <jsp:directive.page contentType="text/html; charset=UTF-8"
7     pageEncoding="UTF-8" session="false" />
8   <jsp:output doctype-root-element="html"
9     doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
10    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
    transitional.dtd"
11    omit-xml-declaration="true" />
12   <![CDATA[<!DOCTYPE html>]]>
13   <html xmlns="http://www.w3.org/1999/xhtml">
14 <c:set var="rootURLPath" value="$
    {pageContext.request.contextPath}"></c:set>
15
16 <jsp:include page="../includes/Header.jspx">
17   <jsp:param name="title" value="Foods R Us!" />
18   <jsp:param name="rootURLPath" value="${rootURLPath}" />
19 </jsp:include>
20
21 <body>
22   <jsp:include page="../includes/Nav.jspx">
23     <jsp:param name="rootURLPath" value="${rootURLPath}" />
24   </jsp:include>
25
26   <fmt:setLocale value = "en_US"/>
27
28   <main class="bd-main">
29   <section class="hero is-small is-primary is-bold">
30     <div class="hero-body">
31       <div class="container">
32         <h1 class="title">Purchase Orders</h1>
33       </div>
34     </div>
35   </section>
36   <div class="container">
37     <section>
38       <c:set var="error" value="${error}" scope="request" />
39       <jsp:include page="../includes/Error.jspx" />
40     </section>
41     <section id="userPOsLinks">
42       <c:if test="${!empty ordersLinks}">
```


PurchaseOrders.jspx

```
43         <c:forEach items="${ordersLinks}" var="order"
varStatus="loop">
44             <p class="is-4 is-link is-size-4"><a
target="_blank" href="${order.value}">Order ${order.key}</a></p>
45         </c:forEach>
46     </c:if>
47 </section>
48 </div>
49 </main>
50
51 <jsp:include page="../includes/Footer.jspx">
52     <jsp:param name="rootURLPath" value="${rootURLPath}" />
53 </jsp:include>
54 </body>
55 </html>
56 </jsp:root>
57
```

Error404.jspx

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
3   xmlns:c="http://java.sun.com/jsp/jstl/core"
4   xmlns:fmt="http://java.sun.com/jsp/jstl/fmt"
5   version="2.0">
6   <jsp:directive.page contentType="text/html; charset=UTF-8"
7     pageEncoding="UTF-8" session="false"/>
8   <jsp:output doctype-root-element="html"
9     doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
10    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
    transitional.dtd"
11    omit-xml-declaration="true" />
12 <![CDATA[<!DOCTYPE html>]]>
13 <html xmlns="http://www.w3.org/1999/xhtml">
14   <jsp:include page="../../includes/Header.jspx">
15     <jsp:param name="title" value="Foods R Us!" />
16     <jsp:param name="rootURLPath" value="${rootURLPath}" />
17   </jsp:include>
18
19   <body>
20     <jsp:include page="../../includes/Nav.jspx">
21       <jsp:param name="rootURLPath" value="${rootURLPath}" />
22     </jsp:include>
23     <main class="bd-main">
24       <section class="hero is-medium is-primary is-bold">
25         <div class="hero-body">
26           <div class="container">
27             <h1 class="title">Food R Us!</h1>
28             <h2 class="subtitle">An eCommerce Site</h2>
29           </div>
30         </div>
31       </section>
32       <div class="container content">
33       </div>
34     </main>
35   </body>
36 </html>
37 </jsp:root>
38
```



JavaScript

main.js

```
1 var appURL = location.href;
2 var appPath = location.pathname;
3 var appRootPath = "/" + appPath.split("/")[1] + "/";
4
5 /**
6  *
7  * Bulma
8  */
9 (function() {
10     let burger = document.querySelector('.burger');
11     let menu = document.querySelector('#'+burger.dataset.target);
12     burger.addEventListener('click', function() {
13         burger.classList.toggle('is-active');
14         menu.classList.toggle('is-active');
15     });
16
17     onClickViewCatItem();
18
19     window.onload = onLoadActions;
20 })();
21
22 /**
23  * On Load Actions
24  *
25  */
26 function onLoadActions() {
27     toggleElemBy('id', 'products-tabs', true);
28 }
29
30
31 /**
32  * Show/Hide and element by id, class or CSS Selector
33  * @param by
34  * @param val
35  * @param hide
36  *
37  */
38 function toggleElemBy(by, val, hide) {
39     let elem = null;
40     if (by == 'id') {
41         elem = document.getElementById(val);
42     } else if (by == 'class') {
43         if (document.getElementsByClassName(val).length > 0)
44         {
```

```
main.js
```

```

45         elem = document.getElementsByClassName(val)[0];
46     }
47 } else if (by == 'css') {
48     elem = document.querySelector(val);
49 }
50
51 if (elem != null) {
52     if (hide == true) {
53         elem.style.display = "none";
54     } else {
55         elem.style.display = "block";
56     }
57 }
58 }
59
60 /**
61  *
62  * Navbar Toggle
63  */
64 document.addEventListener('DOMContentLoaded', () => {
65     // Get all "navbar-burger" elements
66     const $navbarBurgers =
67     Array.prototype.slice.call(document.querySelectorAll('.navbar-
68     burger'), 0);
69
70     // Check if there are any navbar burgers
71     if ($navbarBurgers.length > 0) {
72
73         // Add a click event on each of them
74         $navbarBurgers.forEach( el => {
75             el.addEventListener('click', () => {
76
77                 // Get the target from the "data-target" attribute
78                 const target = el.dataset.target;
79                 const $target = document.getElementById(target);
80
81                 // Toggle the "is-active" class on both the "navbar-
82                 burger" and the
83                 // "navbar-menu"
84                 el.classList.toggle('is-active');
85                 $target.classList.toggle('is-active');
86
87             });
88         });
89     }
90 });

```

main.js

```
86     }
87 });
88
89 /**
90  *
91  * Tabs Toggle
92  */
93 document.querySelectorAll("#tabNav li").forEach(function(navEl) {
94     navEl.onclick = function() { toggleTab(this.id,
95         this.dataset.target); }
96     // toggleTabContent(navEl.dataset.target);
97     if (navEl.className.includes("is-active")) {
98         toggleTab(navEl.id, navEl.dataset.target);
99     }
100 });
101 function toggleTab(selectedNav, targetId) {
102     let navEls = document.querySelectorAll("#tabNav li");
103     navEls.forEach(function(navEl) {
104         if (navEl.id == selectedNav) {
105             navEl.classList.add("is-active");
106         } else {
107             if (navEl.classList.contains("is-active")) {
108                 navEl.classList.remove("is-active");
109             }
110         }
111     });
112
113     toggleTabContent(targetId);
114 }
115
116 function toggleTabContent(targetId) {
117     let tabs = document.querySelectorAll(".tab-pane");
118
119     tabs.forEach(function(tab) {
120         if (tab.id == targetId) {
121             tab.style.display = "block";
122         } else {
123             tab.style.display = "none";
124         }
125     });
126 }
127
128 function hideAllCatsViewAllLink() {
```

main.js

```
129     let viewCatItemsBtns = document.querySelectorAll('a[id^="view-  
cat-item-"]');  
130     viewCatItemsBtns.forEach(function (el) {  
131         toggleElemBy('id', el.id, true);  
132     });  
133 }  
134  
135 function onClickViewCatItem() {  
136     let viewCatItemsBtns = document.querySelectorAll('a[id^="view-  
cat-item-"]');  
137  
138     viewCatItemsBtns.forEach(function (el) {  
139         el.addEventListener('click', () => {  
140  
141             toggleElemBy('id', 'page-products', true);  
142             toggleElemBy('id', 'products-tabs', false);  
143             hideAllCatsViewAllLink();  
144  
145             let catId = el.id.split("view-cat-item-")[1];  
146             document.getElementById("cat-" + catId).click();  
147  
148         });  
149     });  
150 }  
151  
152  
153 function onClickAddToCartBtn(el) {  
154     let productId = el.id.split("product-")[1];  
155     let addToCartUrl = appRootPath + 'Api/Cart';  
156     let qty = el.parentNode.parentNode.querySelector(".qtyInput  
input");  
157  
158     let qs = "addToCart=" + productId + "&" + "qty=" + qty.value;  
159     doSimpleAjax(addToCartUrl, qs, addToCartResult);  
160  
161 }  
162  
163 function addToCartResult(request)  
164 {  
165     // TODO: increase num on top cart btn  
166     let resp = JSON.parse(JSON.stringify(request.responseText));  
167     let checkoutLink =  
document.querySelectorAll("a[href$='Checkout']");  
168
```

main.js

```
169     if (checkoutLink.length < 1) {
170         updatePage();
171     }
172     console.log(resp);
173 }
174
175
176 function updateQty(el) {
177     console.log("updating item qty");
178     let cartUrl = appRootPath + 'Api/Cart';
179     let productNum = el.parentNode.parentNode.id.split('p-')[1];
180     let productQty = document.getElementById("qty-" +
        productNum).value;
181
182     console.log(productNum);
183
184     let qs = "updateQty=" + productNum + "&qty=" + productQty;
185     doSimpleAjax(cartUrl, qs, updateQtyResult);
186 }
187
188 function updateQtyResult(request)
189 {
190     // TODO: increase num on top cart btn
191     console.log('updated qty in cart', request.responseText);
192     updatePage();
193 }
194
195
196 function updateItemsQty(el) {
197     console.log("updating items qty");
198     let cartItemsTable = document.getElementById("cartItems");
199     let items =
        cartItemsTable.querySelector("tbody").getElementsByTagName("tr");
200     let cartUrl = appRootPath + 'Api/Cart';
201
202     for(let i=0; i<items.length; i++) {
203         let item = items[i];
204         let productNum = item.id.split('p-')[1];
205         let productQty = document.getElementById("qty-" +
            productNum).value;
206
207         let qs = "updateQty=" + productNum + "&qty=" + productQty;
208         doSimpleAjax(cartUrl, qs, updateQtyItemsResult);
209     }
```


main.js

```
210
211     updatePage();
212 }
213
214 function updateQtyItemsResult(request)
215 {
216     // TODO: increase num on top cart btn
217     console.log('updated qty in cart', request.responseText);
218 //     updatePage();
219 }
220
221
222 function deleteCartItem(el) {
223     let cartUrl = appRootPath + 'Api/Cart';
224     let productNum = el.parentNode.parentNode.id.split('p-')[1];
225
226     let qs = "removeItem=" + productNum;
227     doSimpleAjax(cartUrl, qs, deleteCartItemResult);
228 }
229
230 function deleteCartItemResult(request)
231 {
232     let req = request.responseText;
233     updatePage();
234
235     if (req != null && req.action == "REMOVE") {
236         let el = document.getElementById('p-' + req.actionReq);
237         console.log(el);
238         console.log(el.parentNode.parentNode);
239         let i = el.parentNode.parentNode.rowIndex;
240         document.getElementById("cartItems").deleteRow(i);
241         console.log('deleted from cart', req);
242     }
243 }
244
245 function doCheckout() {
246     location.href = appRootPath + "Checkout";
247 }
248
249 function updatePage() {
250     location.reload(true);
251 }
252
253 function gotoHomePage() {
```

main.js

```
254     location.href = appRootPath;
255 }
256
257 function doLogin() {
258     location.href = appPath + "?doLogin=";
259 }
260
261 function gotoSearchPg() {
262     let query = document.getElementById("sQuery");
263     let searchParams = "";
264     if (query.value != null && query.value != "") {
265         searchParams = "?query=" + query.value;
266     }
267     location.href = appRootPath + "Search" + searchParams;
268 }
269
270
271 function doSimpleAjax(address, data, handler)
272 {
273     var request = new XMLHttpRequest();
274     request.onreadystatechange = function() {handler(request);};
275     request.open("GET", (address + "?" + data), true);
276     request.send(null);
277 }
278
279
280
281
```



CSS

main.css

```
1
2 html, body {
3     height: 100%;
4     margin: 0;
5 }
6
7 /* Footer – Always stay at bottom */
8 .footer {
9     clear: both;
10    position: relative;
11    z-index: 10;
12    height: 3em;
13    margin-top: 10em;
14 }
15
16 .hero {
17     margin-bottom: 20px;
18 }
19
20 #products-tabs {
21     margin-bottom: 20px;
22 }
23
24 .tab-content {
25     padding: 10px;
26     border-width: 2px !important;
27     border-color: red !important;
28 }
29
30 section {
31     border-bottom: medium solid #f5f5f5;
32 }
33
34
```



Middleware

Main.java

```
1 package Middleware;
2
3
4 import java.util.Scanner;
5
6
7
8 public class Main {
9
10     public static void main(String[] args) throws Exception {
11         String posDir = "";
12
13         // String classPath = System.getProperty("java.class.path");
14         // classPath = classPath.substring(0,
15         // classPath.lastIndexOf(':'));
16         // classPath = classPath.substring(0,
17         // classPath.lastIndexOf('/'));
18
19         String homePath = System.getProperty("user.home");
20         System.out.println(System.getProperty("user.home"));
21         if (args.length > 0) {
22             posDir = args[0];
23         } else {
24             Scanner sc = new Scanner(System.in);
25             System.out.println("Enter Purchase orders directory:");
26
27             posDir = sc.next();
28             sc.close();
29         }
30
31         if (posDir.toLowerCase().equals("exit")) {
32             return;
33         }
34
35         String processedFile = homePath + "/processed.txt";
36         String reportsDir = homePath + "/POReports";
37         // Engine e = Engine.getInstance(posDir, processedFile);
38         String outTo = "json";
39         Engine e = Engine.getInstance(posDir, processedFile,
40         reportsDir, outTo);
41         System.out.println(e.createJsonReport());
42     }
43 }
44 }
```



Model

Engine.java

```
1 package model;
2 import java.io.BufferedReader;
28
29 public class Engine {
30     private static Engine instance = null;
31
32     private static String posDir = System.getProperty("user.home")
+ "/ws_4413/.metadata/.plugins/org.eclipse.wst.server.core/tmp0/
wtpwebapps/eFoods/P0s/";
33     private String processedOrdersFileName;
34     private String reportsDir;
35     private String outTo;
36     private List<Product> products;
37     private List<String> processed;
38     private Report report;
39
40
41     // -----
42     private Engine() throws Exception {
43         this(posDir);
44     }
45
46     private Engine(String P0sDir) throws Exception {
47         this(P0sDir, null);
48     };
49
50     private Engine(String P0sDir, String processedOrdersFileName)
throws Exception {
51         products = new ArrayList<Product>();
52         processed = new ArrayList<String>();
53         report = new Report();
54
55         this.setP0sDir(P0sDir);
56         if (this.checkP0sDir(P0sDir)) {
57             this.processedOrdersFileName =
processedOrdersFileName;
58             this.getProcessedFromFile(processedOrdersFileName);
59
60             processP0s(posDir);
61             report.setProducts(products);
62             this.saveProcessedToFile();
63         }
64     }
65
```


Engine.java

```
66     private Engine(String P0sDir, String processedOrdersFileName,
String reportsDir, String outTo) throws Exception {
67         products = new ArrayList<Product>();
68         processed = new ArrayList<String>();
69         report = new Report();
70
71         this.setP0sDir(P0sDir);
72         this.reportsDir = reportsDir;
73         this.outTo = outTo;
74
75         if (this.checkP0sDir(P0sDir)) {
76             this.processedOrdersFileName =
processedOrdersFileName;
77             this.getProcessedFromFile(processedOrdersFileName);
78
79             processP0s(posDir);
80             report.setProducts(products);
81             this.saveProcessedToFile();
82             this.saveReportToFile(outTo);
83         }
84     }
85
86
87     // -----
88     public static Engine getInstance(String P0sDir, String
processedOrdersFileName, String reportsDir, String outTo) throws
Exception {
89         if (instance == null) {
90             instance = new Engine(P0sDir, processedOrdersFileName,
reportsDir, outTo);
91         }
92         return instance;
93     }
94
95     public static Engine getInstance(String P0sDir, String
processedOrdersFileName) throws Exception {
96         if (instance == null) {
97             instance = new Engine(P0sDir,
processedOrdersFileName);
98         }
99         return instance;
100     }
101
102     public static Engine getInstance(String P0sDir) throws
```

Engine.java

```
Exception {
103     return getInstance(P0sDir, null);
104 }
105
106 public static Engine getInstance() throws Exception {
107     return getInstance(posDir);
108 }
109
110
111 // -----
112 public List<String> getProcessed() {
113     return processed;
114 }
115
116 public List<Product> getProducts() {
117     return products;
118 }
119
120 public void setP0sDir(String P0sDir) {
121     if (P0sDir.equals("null") || P0sDir.equals("none")) {
122         posDir = posDir;
123     } else if (checkP0sDir(P0sDir) ) {
124         posDir = P0sDir;
125     }
126 }
127
128 public boolean checkP0sDir(String P0sDir) {
129     if (P0sDir != null && !P0sDir.isEmpty()) {
130         return true;
131     } else if (P0sDir.equals("null") || P0sDir.equals("none"))
132 {
133     return true;
134 }
135     return false;
136 }
137
138 // -----
139 public synchronized void processP0s(File[] files) throws
Exception {
140     for (File f: files) {
141         processP0(f);
142     }
143 }
```

Engine.java

```
144
145     public synchronized void processP0s(String dirName) throws
Exception {
146         processP0s(new File(dirName));
147     }
148
149     public synchronized void processP0s(File dir) throws Exception
{
150         processP0s(dir.listFiles());
151     }
152
153
154     // -----
155     public synchronized void processP0(File f, List<Product>
products, List<String> processed) throws Exception {
156         if (getFileExtension(f).equals(".xml")) {
157             if (!processed.contains(f.getName())) {
158
159                 DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
160                 DocumentBuilder dBuilder =
dbFactory.newDocumentBuilder();
161                 Document doc = dBuilder.parse(f);
162                 doc.getDocumentElement().normalize();
163
164                 NodeList items = doc.getElementsByTagName("item");
165                 for (int i=0; i<items.getLength(); i++) {
166                     Node item = items.item(i);
167                     if (item.getNodeType() == Node.ELEMENT_NODE) {
168
169                         Element e = (Element) item;
170
171                         String itemNum = e.getAttribute("number");
172                         String itemName =
e.getElementsByTagName("name").item(0).getTextContent();
173                         String itemQtyVal =
e.getElementsByTagName("quantity").item(0).getTextContent();
174
175                         if (itemNum != null && itemNum != "") {
176                             int itemQty =
Integer.parseInt(itemQtyVal);
177
178                             Product p = new Product(itemNum,
itemName, itemQty);
```

Engine.java

```
179             if (!products.contains(p)) {
180                 products.add(p);
181             } else {
182                 p =
products.get(products.indexOf(p));
183                 p.increaseQty(itemQty);
184                 products.set(products.indexOf(p),
p);
185             }
186         }
187     }
188 }
189
190         processed.add(f.getName());
191     }
192 }
193 }
194
195     public synchronized void processP0(File f) throws Exception {
196         processP0(f, this.products, this.processed);
197     }
198
199
200     // -----
201     public synchronized JsonObject createJsonReport(List<Product>
products) throws Exception {
202         JsonObject json = new JsonObject();
203         JsonElement jsonElement = new Gson().toJsonTree(report);
204         json.add("data", jsonElement);
205         return json;
206     }
207
208     public synchronized JsonObject createJsonReport() throws
Exception {
209         return createJsonReport(this.products);
210     }
211
212
213     // -----
214     public synchronized String createXmlReport(List<Product>
products) throws Exception {
215         JAXBContext jc = JAXBContext.newInstance(Report.class);
216         Marshaller m = jc.createMarshaller();
217         StringWriter sw = new StringWriter();
```

Engine.java

```
218         m.marshal(new Report(), sw);
219         return sw.toString();
220     }
221
222     public synchronized String createXmlReport() throws Exception
223     {
224         return createXmlReport(this.products);
225     }
226
227     public synchronized void saveReportToFile(String outTo) throws
228     Exception {
229         this.checkDir(this.reportsDir);
230         String reportFile =
231         (this.reportsDir.charAt(this.reportsDir.length()-1) == '/') ?
232         this.reportsDir : this.reportsDir + "/";
233         reportFile = reportFile + report.getGeneratedOn();
234
235         String reportCreated = "";
236
237         if (outTo.toLowerCase().equals("xml")) {
238             reportFile = reportFile + ".xml";
239             reportCreated = this.createXmlReport();
240         } else if (outTo.toLowerCase().equals("json")) {
241             reportFile = reportFile + ".json";
242             reportCreated = this.createJsonReport().toString();
243         }
244         FileWriter fw = new FileWriter(reportFile);
245         fw.write(reportCreated);
246         fw.close();
247     }
248
249     // -----
250     public synchronized List<String> getProcessedFromFile(String
251     filename) throws Exception {
252         File f = this.checkProcessedOrdersFile(filename);
253         BufferedReader br = new BufferedReader(new FileReader(f));
254         String line;
255         while ((line = br.readLine()) != null) {
256             if (!processed.contains(line)) {
257                 processed.add(line);
258             }
259         }
260         br.close();
261     }
```

Engine.java

```
257         return processed;
258     }
259
260     public synchronized List<String> getProcessedFromFile() throws
Exception {
261         return
this.getProcessedFromFile(this.processedOrdersFileName);
262     }
263
264
265     // -----
266     public synchronized void saveProcessedToFile(String filename)
throws Exception {
267         File f = this.checkProcessedOrdersFile(filename);
268         FileWriter fw = new FileWriter(f);
269         for (String str: processed) {
270             if (!str.isEmpty() && str != null) {
271                 fw.write(str + "\n");
272             }
273         }
274         fw.close();
275     }
276
277     public synchronized void saveProcessedToFile() throws
Exception {
278         saveProcessedToFile(this.processedOrdersFileName);
279     }
280
281
282     // -----
283     public synchronized File checkProcessedOrdersFile(String
filename) throws Exception {
284         File f = new File(filename);
285         if (!f.exists()) {
286             f.createNewFile();
287         }
288         return f;
289     }
290
291     public synchronized File checkDir(String dirName) throws
Exception {
292         File f = new File(dirName);
293         if (!f.exists()) {
294             f.mkdir();
```

Engine.java

```
295         }
296         return f;
297     }
298
299     public synchronized File checkProcessedOrdersFile() throws
Exception {
300         return
checkProcessedOrdersFile(this.processedOrdersFileName);
301     }
302
303
304     // -----
305     private synchronized static String getFileExtension(File file)
{
306         String name = file.getName();
307         int lastIndexOf = name.lastIndexOf(".");
308         if (lastIndexOf == -1) {
309             return ""; // empty extension
310         }
311         return name.substring(lastIndexOf);
312     }
313
314     public synchronized void clearProducts() {
315         products.clear();
316     }
317
318     public synchronized void clearProcessed() {
319         processed.clear();
320     }
321
322
323 }
324
```

Report.java

```
1 package model;
2 import java.util.ArrayList;
9
10 @XmlElement(name="report")
11 public class Report {
12     private List<Product> products;
13     private String generatedOn;
14
15     public Report(List<Product> products) {
16         this.products = products;
17         this.generatedOn = new Date().toString();
18     }
19
20     public Report() {
21         this(new ArrayList<Product>());
22     }
23
24     @XmlElementWrapper(name="products")
25     @XmlElement(name="product")
26     public List<Product> getProducts() {
27         return products;
28     }
29
30     public void setProducts(List<Product> products) {
31         this.products = products;
32     }
33
34     public String getGeneratedOn() {
35         return generatedOn;
36     }
37
38     public void setGeneratedOn(String generatedOn) {
39         this.generatedOn = generatedOn;
40     }
41
42     @Override
43     public String toString() {
44         return this.products.toString();
45     }
46 }
47
```


Product.java

```
1 package model;
2 import javax.xml.bind.annotation.XmlAttribute;
3
4
5
6 @XmlRootElement(name="item")
7 public class Product {
8     private String number;
9     private String name;
10    private int qty;
11
12    public Product() {
13    }
14
15    public Product(String number, String name, int qty) {
16        this.setNumber(number);
17        this.setName(name);
18        this.setQty(qty);
19    }
20
21    @XmlAttribute(name="number")
22    public String getNumber() {
23        return number;
24    }
25
26    public void setNumber(String number) {
27        this.number = number;
28    }
29
30    @XmlElement(name="name")
31    public String getName() {
32        return name;
33    }
34
35    public void setName(String name) {
36        this.name = name;
37    }
38
39    @XmlElement(name="quantity")
40    public int getQty() {
41        return qty;
42    }
43
44    public void setQty(int qty) {
45        this.qty = qty;
46    }
47 }
```

Product.java

```
47
48     public synchronized void increaseQty() {
49         this.qty++;
50     }
51
52     public synchronized void increaseQty(int qty) {
53         this.qty += qty;
54     }
55
56     @Override
57     public String toString() {
58         return String.format("{ Number: %s | Name: %s | Quantity:
59 %s }\n", number, name, qty);
60     }
61
62     @Override
63     public boolean equals(Object obj) {
64         if (this == obj) return true;
65         if (obj == null) return false;
66         if (!(obj instanceof Product)) return false;
67
68         Product other = (Product) obj;
69         if (!this.number.equals(other.number)) {
70             return false;
71         }
72         return true;
73     }
74 }
75
```



Helpers

Utils.java

```
1 package helpers;
2
3 import java.io.InputStream;
23
24
25 public class Utils {
26
27     public static void renderXsltTHtml(String xml, InputStream
    xslIs, Writer out) throws Exception {
28         DocumentBuilder factory_db =
    DocumentBuilderFactory.newInstance().newDocumentBuilder();
29         Document doc = factory_db.parse(new InputSource(new
    StringReader(xml)));
30         Source xmlSource = new DOMSource(doc);
31
32         TransformerFactory transformerFactory =
    TransformerFactory.newInstance();
33         Templates cachedXSLT = transformerFactory.newTemplates(new
    StreamSource(xslIs));
34         Transformer transformer = cachedXSLT.newTransformer();
35
36         transformer.transform(xmlSource, new StreamResult(out));
37     }
38
39     public static double round(double value, int places) {
40         if (places < 0) throw new IllegalArgumentException();
41
42         BigDecimal bd = new BigDecimal(value);
43         bd = bd.setScale(places, RoundingMode.HALF_UP);
44         return bd.doubleValue();
45     }
46 }
47
```