

```
1: <?php
2: session_start();
3:
4: $CONFIG_FILE = "../config/config.json";
5:
6: $config = get_config($CONFIG_FILE);
7: $alerts = array();
8:
9: // Functions & Helpers
10: function get_config($f) {
11:     return json_decode(file_get_contents($f));
12: }
13:
14: function do_auth($username, $password, $challenge_url, $challenge_file) {
15:     $ch = curl_init();
16:     curl_setopt($ch, CURLOPT_URL, $challenge_url);
17:     curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
18:     curl_setopt($ch, CURLOPT_VERBOSE, true);
19:     curl_setopt($ch, CURLOPT_TIMEOUT, 5);
20:     curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 10);
21:     curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
22:     curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
23:     curl_setopt($ch, CURLOPT_USERPWD, "$username:$password");
24:     curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
25:     $output = trim(curl_exec($ch));
26:     $info = curl_getinfo($ch);
27:     curl_close($ch);
28:     error_log(print_r($info, true));
29:     if ($info["http_code"] == 403 || $info["http_code"] == 401) {
30:         return array("message" => "Invalid Username or Password.");
31:     } else if ($info["http_code"] != 200) {
32:         return array("message" => "Authentication service is experiencing problems at
the moment. Please try again later.");
33:     }
34:
35:     $challenge = trim(file_get_contents($challenge_file));
36:
37:     if (empty($challenge) || empty($output)) {
38:         return array("message" => "Authentication service is not configured properly."
);
39:     }
40:
41:     if ($challenge != $output) {
42:         return array("message" => "Authentication service is not active at the moment.
");
43:     }
44:
45:     return true;
46: }
47:
48: function validate_payload($raw_payload, $raw_signature, $shared_key) {
49:     $remote_payload = base64_decode($_REQUEST['payload']);
50:     $signature = hash_hmac('sha256', $_REQUEST['payload'], $shared_key);
51:     if (strtolower($signature) == strtolower($raw_signature)) {
52:         parse_str($remote_payload, $payload_array);
53:         return $payload_array;
54:     } else {
55:         return false;
56:     }
57: }
58:
59: function get_redirect_url($base_url, $nonce, $data, $shared_key) {
60:     $data["nonce"] = $nonce;
61:     $raw_payload = http_build_query($data);
62:     $payload = base64_encode($raw_payload);
63:     $signature = strtolower(hash_hmac('sha256', $payload, $shared_key));
64:     $query_string = "payload=" . urlencode($payload) . "&signature=" . urlencode($si
gnature);
65:     return $base_url . "?" . $query_string;
66: }
```

```
67:
68: function get_full_name($username) {
69:     $system_name = exec('getent passwd ' . escapeshellarg($username) . ' | cut -d: -
f5 | cut -d, -f1');
70:     return empty($system_name) ? "Unknown Name" : $system_name;
71: }
72: // End functions & helpers
73:
74: if (!empty($_REQUEST['submit'])) {
75:     $payload = validate_payload($_REQUEST['payload'], $_REQUEST['signature'], $confi
g->sharedKey);
76:     $username = $_REQUEST['username'];
77:     $password = $_REQUEST['password'];
78:
79:     if (!$payload === false)
80:         $authenticate = do_auth($username,$password,$config->challengeURL, $config->ch
allengeFile);
81:     else
82:         $authenticate = array("message" => "Login request is expired. Please go back t
o the site and click long again.");
83:
84:     if ($authenticate === true) {
85:         http_response_code(200);
86:         $data = array();
87:         $data["username"] = $username;
88:         $data["fullname"] = get_full_name($username);
89:
90:         //Testing
91:         $ALT_REDIRECT = empty($_SESSION["REDIRECT_TO"]) ? $config->redirectURL : $_SES
SION["REDIRECT_TO"];
92:         //End testing
93:         error_log("Redirect To : " . $ALT_REDIRECT);
94:
95:         $url = get_redirect_url($ALT_REDIRECT, $payload["nonce"], $data, $config->shar
edKey);
96:         header('Location: ' . $url);
97:         die();
98:     } else {
99:         http_response_code(403);
100:         array_push($alerts, $authenticate["message"]);
101:     }
102: } else {
103:     if (empty($_REQUEST['payload']) || empty($_REQUEST['signature'])) {
104:         $error = "Please provide a valid payload and a signature to use the authentica
tion service.";
105:     } else {
106:         $payload = validate_payload($_REQUEST['payload'], $_REQUEST['signature'], $con
fig->sharedKey);
107:         if ($payload === false) {
108:             $error = "Login request is expired. Please go back to the site and click log
in again.";
109:         }
110:     }
111:     if (!empty($_SERVER["HTTP_REFERER"]) && !empty($_REQUEST["redirect"])) {
112:         $_SESSION["REDIRECT_TO"] = $_SERVER["HTTP_REFERER"] . $_REQUEST["redirect"];
113:     }
114: }
115: ?>
116: <!doctype html>
117: <html class="no-js">
118: <head>
119:     <meta charset="utf-8">
120:     <title>Auth</title>
121:     <meta name="description" content="">
122:     <meta name="viewport" content="width=device-width">
123:     <link rel="shortcut icon" href="/favicon.ico">
124:     <!-- Place favicon.ico and apple-touch-icon.png in the root directory -->
125:     <!-- build:css(.) styles/vendor.css -->
126:     <!-- bower:css -->
127:     <!-- endbower -->
```

```
128: <!-- endbuild -->
129: <!-- build:css(.tmp) styles/main.css -->
130: <link rel="stylesheet" href="styles/main.css">
131: <!-- endbuild -->
132: <!-- build:js scripts/vendor/modernizr.js -->
133: <!-- endbuild -->
134:
135: </head>
136: <body>
137:     <!--[if lt IE 10]>
138:         <p class="browsehappyp">You are using an <strong>outdated</strong> browser. P
139: lease <a href="http://browsehappyp.com/">upgrade your browser</a> to improve your experien
140: ce.</p>
141:     <![endif]-->
142:     <div class="container">
143:         <div class="header">
144:             <ul class="nav nav-pills pull-right">
145:                 <li class="active"><a href="#">Authenticate</a></li>
146:                 <li><a href="#">About</a></li>
147:             </ul>
148:             <h3 class="text-muted">Lassonde Auth</h3>
149:         </div>
150:         <div class="alert-container">
151:             <?php
152:                 if (!empty($alerts)) {
153:                     foreach($alerts as $alert) {
154:                         ?>
155:                         <div class="alert alert-danger" role="alert">
156:                             <span class="glyphicon glyphicon-exclamation-sign" aria-hidden="tr
157: ue"></span>
158:                             <span class="sr-only">Error:</span>
159:                             <?php echo $alert ?>
160:                         </div>
161:                         <?php
162:                     }
163:                 ?>
164:             </div>
165:             <div class="jumbotron">
166:                 <?php if (empty($error)) { ?>
167:                 <form role="form" id="login-form" action="" method="POST">
168:                 <div class="form-group">
169:                     <div class="input-group input-group-lg">
170:                         <span class="input-group-addon">@</span>
171:                         <input required type="text" name="username" class="form-control" pla
172: ceholder="Username">
173:                     </div>
174:                 </div>
175:                 <div class="form-group">
176:                     <div class="input-group input-group-lg">
177:                         <span class="input-group-addon">@</span>
178:                         <input required type="password" name="password" class="form-control"
179: placeholder="Password">
180:                     </div>
181:                 </div>
182:                 <input type="hidden" name="submit" value="true"/>
183:                 <input type="hidden" name="payload" value="<?php echo $_REQUEST["payload"]
184: ?>"/>
185:                 <input type="hidden" name="signature" value="<?php echo $_REQUEST["signatu
186: re"] ?>"/>
187:                 <p class="lead">You will be authenticated against EECS LDAP.</p>
188:                 <input type="submit" name="submit" value="Login" class="btn btn-lg btn
189: -primary"/>
190:                 <br/><br/>
191:             </form>
192:             <?php } else { ?>
193:             <p class="h2 text-center">Error</p>
```

```
190:         <p class="h4 text-center"><?php echo htmlspecialchars($error) ?></p>
191:         <?php } ?>
192:     </div>
193:
194:     <div class="row marketing">
195:         <div class="col-lg-4">
196:             <h4>What is this?</h4>
197:             <p>This authenticates.</p>
198:         </div>
199:         <div class="col-lg-4">
200:             <h4>What is this?</h4>
201:             <p>This authenticates.</p>
202:         </div>
203:         <div class="col-lg-4">
204:             <h4>What is this?</h4>
205:             <p>This authenticates.</p>
206:         </div>
207:     </div>
208:
209:     <div class="footer">
210:         <p><span class="glyphicon glyphicon-heart"></span> from the Yeoman team<
/p>
211:     </div>
212:
213: </div>
214:
215:
216: <!-- build:js(.) scripts/vendor.js -->
217: <!-- bower:js -->
218: <!-- endbower -->
219: <!-- endbuild -->
220:
221: </body>
222: </html>
```

```

1: package b2b;
2:
3: import java.io.File;
4: import java.util.ArrayList;
5: import java.util.Iterator;
6: import java.util.List;
7: import java.util.Map;
8: import java.util.TreeMap;
9:
10: import javax.xml.bind.JAXBContext;
11: import javax.xml.bind.Unmarshaller;
12:
13: import beans.AggregItemBean;
14: import beans.AggregOrderBean;
15: import beans.ItemBean;
16: import beans.POBan;
17:
18: // class that reads and consolidates purchase orders
19: public class Aggregator
20: {
21:     public Aggregator()
22:     {
23:     }
24:
25:     public AggregOrderBean getAggregatedOrder(String[] fnames, int id) throws Exc
ption
26:     {
27:         // get purchase orders
28:         List<POBan> orders = this.getOrders(fnames);
29:
30:         // map that temporally stores item list
31:         Map<String, AggregItemBean> mp = new TreeMap<String, AggregItemBean>();
32:
33:         // aggregate orders
34:         Iterator<POBan> itr = orders.iterator();
35:         while (itr.hasNext())
36:         {
37:             List<ItemBean> items = itr.next().getItems().getItemsList(
);
38:
39:             // loop over items in a single order
40:             Iterator<ItemBean> itr2 = items.iterator();
41:             while (itr2.hasNext())
42:             {
43:                 ItemBean item = itr2.next();
44:                 String key = item.getNumber();
45:
46:                 if (mp.containsKey(key))
47:                 {
48:                     mp.get(key).addQuantity(item.getQuantity()
);
49:                 }
50:                 else
51:                 {
52:                     mp.put(key, new AggregItemBean(item));
53:                 }
54:             }
55:         } // end of aggregating orders into map
56:
57:         // store aggregated order into AggregOrderBean
58:         List<AggregItemBean> aggItems = new ArrayList<AggregItemBean>();
59:         Iterator<AggregItemBean> itr3 = mp.values().iterator();
60:         while (itr3.hasNext())
61:         {
62:             aggItems.add(itr3.next());
63:         }
64:
65:         return new AggregOrderBean(aggItems, id);
66:     }
67:

```

```

68:         private List<POBean> getOrders(String[] fnames) throws Exception
69:         {
70:             if (fnames == null)
71:                 throw new NullPointerException("invalid file names.");
72:
73:             // define unmarshaller
74:             JAXBContext jc;
75:             Unmarshaller um;
76:             try
77:             {
78:                 jc = JAXBContext.newInstance(POBean.class);
79:                 um = jc.createUnmarshaller();
80:             }
81:             catch (Exception e)
82:             {
83:                 throw new Exception("Error occurs when defining unmarshall
er.");
84:             }
85:
86:             // unmarshal order files
87:             List<POBean> orders = new ArrayList<POBean>();
88:             for (int i = 0; i < fnames.length; i++)
89:             {
90:                 try
91:                 {
92:                     orders.add((POBean) um.unmarshal(new File(fnames[i
])));
93:                 }
94:                 catch (Exception e)
95:                 {
96:                     throw new Exception("Error occurs during unmarshall
ling.");
97:                 }
98:             }
99:
100:             return orders;
101:         }
102:     }

```

```

1: package b2b;
2:
3: import java.io.File;
4: import java.util.ArrayList;
5: import java.util.Iterator;
6: import java.util.List;
7:
8: import javax.xml.bind.JAXBContext;
9: import javax.xml.bind.Marshaller;
10:
11: import beans.AggregItemBean;
12: import beans.AggregOrderBean;
13:
14: public class Purchaser
15: {
16:     private List<Wholesaler> wholesalers;
17:     private String key;
18:
19:     public Purchaser()
20:     {
21:         this.wholesalers = new ArrayList<Wholesaler>();
22:     }
23:
24:     public void purchase(AggregOrderBean order, String outfile) throws Exception
25:     {
26:         Iterator<AggregItemBean> itr = order.getItems().iterator();
27:         while (itr.hasNext())
28:         {
29:             AggregItemBean item = itr.next();
30:
31:             // find best offer
32:             Wholesaler best = null;
33:             double bestOffer = Double.POSITIVE_INFINITY;
34:             Iterator<Wholesaler> itr2 = this.wholesalers.iterator();
35:             while (itr2.hasNext())
36:             {
37:                 Wholesaler cur = itr2.next();
38:                 double offer = cur.quote(item.getNumber());
39:                 if (offer >= 0 && offer < bestOffer)
40:                 {
41:                     best = cur;
42:                     bestOffer = offer;
43:                 }
44:             } // end of iterating wholesalers
45:
46:             // item out of stock
47:             if (best == null)
48:             {
49:                 item.setPrice(-1.0);
50:                 item.setWholesaler("No wholesaler");
51:                 item.setCode("Out of stock");
52:                 item.setOrdered(false);
53:             }
54:             else // take the best offer
55:             {
56:                 String code;
57:                 try
58:                 {
59:                     code = best.order(item.getNumber(), item.getQuantity(), this.key);
60:                     if (code.equals("Invalid key!"))
61:                     {
62:                         item.setOrdered(false);
63:                     }
64:                     else
65:                     {
66:                         item.setOrdered(true);
67:                     }
68:                     item.setPrice(bestOffer);
69:                     item.setCode(code);

```

```
70:                                     item.setWholesaler(best.getName());
71:                                     }
72:                                     catch (Exception e)
73:                                     {
74:                                         item.setOrdered(false);
75:                                     }
76:                                     }
77:                                     } // end of iterating items
78:
79:                                     // generate report
80:                                     this.generateReport(order, outfile);
81:                                     }
82:
83:     public boolean addWholesaler(Wholesaler ws)
84:     {
85:         return this.wholesalers.add(ws);
86:     }
87:
88:     public void setKey(String key)
89:     {
90:         this.key = key;
91:     }
92:
93:     private void generateReport(AggOrderBean order, String filename) throws Ex
ception
94:     {
95:         // marshaling
96:         try
97:         {
98:             JAXBContext jc = JAXBContext.newInstance(order.getClass())
;
99:             Marshaller mar = jc.createMarshaller();
100:             mar.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
TRUE);
101:             mar.setProperty(Marshaller.JAXB_FRAGMENT, Boolean.TRUE);
102:             mar.marshal(order, new File(filename));
103:         }
104:         catch (Exception e)
105:         {
106:             throw new Exception("Error occurs during marshalling.");
107:         }
108:     }
109: }
```



```

1: package b2b;
2:
3: import java.net.URL;
4:
5: import javax.xml.soap.*;
6:
7: import org.w3c.dom.NodeList;
8:
9: public class Wholesaler
10: {
11:     private String name;
12:     private URL url;
13:
14:     public Wholesaler(String name, URL url)
15:     {
16:         this.name = name;
17:         this.url = url;
18:     }
19:
20:     public String getName()
21:     {
22:         return name;
23:     }
24:
25:     public void setName(String name)
26:     {
27:         this.name = name;
28:     }
29:
30:     public URL getUrl()
31:     {
32:         return url;
33:     }
34:
35:     public void setUrl(URL url)
36:     {
37:         this.url = url;
38:     }
39:
40:     public double quote(String itemNum) throws Exception
41:     {
42:         String[] params = {"itemNumber"};
43:         String[] values = {itemNum};
44:
45:         return Double.parseDouble(this.operation("quote", params, values).
item(0).getTextContent());
46:     }
47:
48:     public String order(String itemNum, int quantity, String key) throws Excep
tion
49:     {
50:         String[] params = {"itemNumber", "quantity", "key"};
51:         String[] values = {itemNum, String.valueOf(quantity), key};
52:
53:         return this.operation("order", params, values).item(0).getTextCont
ent();
54:     }
55:
56:     private NodeList operation(String operation, String[] reqParams, String[]
values) throws Exception
57:     {
58:         // create SOAP message
59:         SOAPMessage msg;
60:         try
61:         {
62:             msg = MessageFactory.newInstance().createMessage();
63:
64:             MimeHeaders header = msg.getMimeHeaders();
65:             header.addHeader("SOAPAction", "");
66:

```

```

67:         SOAPBody body = msg.getSOAPPart().getEnvelope().getBody();
68:         SOAPElement elm = body.addChildElement(operation);
69:         for (int i=0; i < reqParams.length; i++)
70:         {
71:             elm.addChildElement(reqParams[i]).addTextNode(valu
es[i]);
72:         }
73:     }
74:     catch (Exception e)
75:     {
76:         throw new Exception("Error occurs when creating SOAP messa
ge.");
77:     }
78:
79:     // connect to the wholesaler web service
80:     SOAPMessage resp;
81:     try
82:     {
83:         SOAPConnection sc = SOAPConnectionFactory.newInstance().cr
eateConnection();
84:         resp = sc.call(msg, this.url);
85:         sc.close();
86:     }
87:     catch (Exception e)
88:     {
89:         throw new Exception("Error occurs when setting up SOAP con
nection.");
90:     }
91:
92:     // parse the response
93:     NodeList result;
94:     try
95:     {
96:         result = resp.getSOAPPart().getEnvelope().getBody().getEle
mentsByTagName(operation + "Return");
97:         // resp.writeTo(System.out);
98:     }
99:     catch (Exception e)
100:     {
101:         throw new Exception("Error occurs when parsing SOAP return
.");
102:     }
103:
104:     return result;
105: }
106: }

```

```
1: package beans;
2:
3: import javax.xml.bind.annotation.XmlAttribute;
4: import javax.xml.bind.annotation.XmlElement;
5: import javax.xml.bind.annotation.XmlRootElement;
6:
7: // item bean for aggregated order
8: @XmlRootElement
9: public class AggItemBean
10: {
11:     // @XmlAttribute
12:     private String number;
13:     // @XmlElement
14:     private String name;
15:     // @XmlElement
16:     private int quantity;
17:
18:     // the following attributes may not be defined at creation
19:     // @XmlElement
20:     private double price;
21:     // @XmlElement
22:     private double extended;
23:     // @XmlElement
24:     private String wholesaler;
25:     // @XmlElement
26:     private String code;
27:     // @XmlElement
28:     private boolean ordered;
29:
30:     public AggItemBean()
31:     {
32:         super();
33:     }
34:
35:     public AggItemBean(ItemBean item)
36:     {
37:         this.number = item.getNumber();
38:         this.name = item.getName();
39:         this.quantity = item.getQuantity();
40:         this.ordered = false;
41:     }
42:
43:     public AggItemBean(String number, String name, double price, int quantity,
44:         String wholesaler, String code)
45:     {
46:         this.number = number;
47:         this.name = name;
48:         this.price = price;
49:         this.quantity = quantity;
50:         this.extended = 0;
51:         this.wholesaler = wholesaler;
52:         this.code = code;
53:         this.ordered = false;
54:     }
55:
56:     public void addQuantity(int amount)
57:     {
58:         this.quantity = this.quantity + amount;
59:     }
60:
61:     // ===== getters and setters =====
62:     @XmlAttribute
63:     public String getNumber()
64:     {
65:         return number;
66:     }
67:
68:     public void setNumber(String number)
69:     {
70:         this.number = number;
```

```
71:         }
72:
73:         @XmlElement
74:         public String getName()
75:         {
76:             return name;
77:         }
78:
79:         public void setName(String name)
80:         {
81:             this.name = name;
82:         }
83:
84:         @XmlElement
85:         public double getPrice()
86:         {
87:             return price;
88:         }
89:
90:         public void setPrice(double price)
91:         {
92:             this.price = price;
93:         }
94:
95:         @XmlElement
96:         public int getQuantity()
97:         {
98:             return quantity;
99:         }
100:
101:         public void setQuantity(int quantity)
102:         {
103:             this.quantity = quantity;
104:         }
105:
106:         @XmlElement
107:         public double getExtended()
108:         {
109:             this.updateExtended();
110:             return extended;
111:         }
112:
113:         @XmlElement
114:         public String getWholesaler()
115:         {
116:             return wholesaler;
117:         }
118:
119:         public void setWholesaler(String wholesaler)
120:         {
121:             this.wholesaler = wholesaler;
122:         }
123:
124:         @XmlElement
125:         public String getCode()
126:         {
127:             return code;
128:         }
129:
130:         public void setCode(String code)
131:         {
132:             this.code = code;
133:         }
134:
135:         @XmlElement
136:         public boolean isOrdered()
137:         {
138:             return ordered;
139:         }
140:
```

```
141:      public void setOrdered(boolean ordered)
142:      {
143:          this.ordered = ordered;
144:      }
145:
146:      private void updateExtended()
147:      {
148:          if (this.ordered)
149:          {
150:              this.extended = this.price * this.quantity;
151:          }
152:          else
153:          {
154:              this.extended = 0;
155:          }
156:      }
157: }
```

```
1: package beans;
2:
3: import java.util.Iterator;
4: import java.util.List;
5:
6: import javax.xml.bind.annotation.XmlAttribute;
7: import javax.xml.bind.annotation.XmlElement;
8: import javax.xml.bind.annotation.XmlRootElement;
9:
10: // aggregated order bean
11: @XmlRootElement(name="procurement")
12: public class AggOrderBean
13: {
14:     // @XmlElement(name="item")
15:     private List<AggItemBean> items;
16:     // @XmlElement
17:     private double total;
18:     // @XmlAttribute
19:     private int id;
20:
21:     public AggOrderBean()
22:     {
23:         super();
24:     }
25:
26:     public AggOrderBean(List<AggItemBean> items, int id)
27:     {
28:         this.items = items;
29:         this.updateTotal();
30:         this.id = id;
31:     }
32:
33:     // ===== getters and setters =====
34:     @XmlElement(name="item")
35:     public List<AggItemBean> getItems()
36:     {
37:         return items;
38:     }
39:
40:     public void setItems(List<AggItemBean> items)
41:     {
42:         this.items = items;
43:     }
44:
45:     @XmlElement
46:     public double getTotal()
47:     {
48:         this.updateTotal();
49:         return total;
50:     }
51:
52:     @XmlAttribute
53:     public int getId()
54:     {
55:         return id;
56:     }
57:
58:     public void setId(int id)
59:     {
60:         this.id = id;
61:     }
62:
63:     private void updateTotal()
64:     {
65:         this.total = 0;
66:         Iterator<AggItemBean> itr = this.items.iterator();
67:         while (itr.hasNext())
68:         {
69:             this.total = this.total + itr.next().getExtended();
70:         }
71:     }
72: }
```

```
71:         }  
72:     }
```

```
1: package beans;
2:
3: import javax.xml.bind.annotation.XmlAttribute;
4: import javax.xml.bind.annotation.XmlElement;
5: import javax.xml.bind.annotation.XmlRootElement;
6:
7: @XmlRootElement
8: public class CustomerBean
9: {
10:     // @XmlElement
11:     private String name;
12:     // @XmlAttribute
13:     private String account;
14:
15:     public CustomerBean()
16:     {
17:         super();
18:     }
19:
20:     public CustomerBean(String name, String account)
21:     {
22:         this.name = name;
23:         this.account = account;
24:     }
25:
26:     // ===== getters and setters =====
27:     @XmlElement
28:     public String getName()
29:     {
30:         return name;
31:     }
32:
33:     public void setName(String name)
34:     {
35:         this.name = name;
36:     }
37:
38:     @XmlAttribute
39:     public String getAccount()
40:     {
41:         return account;
42:     }
43:
44:     public void setAccount(String account)
45:     {
46:         this.account = account;
47:     }
48: }
```



```

1: package beans;
2:
3: import javax.xml.bind.annotation.XmlAttribute;
4: import javax.xml.bind.annotation.XmlElement;
5: import javax.xml.bind.annotation.XmlRootElement;
6:
7: @XmlRootElement
8: public class ItemBean
9: {
10:     // @XmlElement
11:     private String name;
12:     // @XmlElement
13:     private double price;
14:     // @XmlElement
15:     private int quantity;
16:     // @XmlElement
17:     private double extended;           // = price * quantity
18:     // @XmlAttribute
19:     private String number; // item number
20:
21:     public ItemBean()
22:     {
23:         super();
24:     }
25:
26:     public ItemBean(String name, double price, int quantity, double extended,
27:         String number)
28:     {
29:         this.name = name;
30:         this.price = price;
31:         this.quantity = quantity;
32:         this.extended = extended;
33:         this.number = number;
34:     }
35:
36:     // ===== getters and setters =====
37:     @XmlElement
38:     public String getName()
39:     {
40:         return name;
41:     }
42:
43:     public void setName(String name)
44:     {
45:         this.name = name;
46:     }
47:
48:     @XmlElement
49:     public double getPrice()
50:     {
51:         return price;
52:     }
53:
54:     public void setPrice(double price)
55:     {
56:         this.price = price;
57:     }
58:
59:     @XmlElement
60:     public int getQuantity()
61:     {
62:         return quantity;
63:     }
64:
65:     public void setQuantity(int quantity)
66:     {
67:         this.quantity = quantity;
68:     }
69:
70:     @XmlElement

```

```
71:      public double getExtended()
72:      {
73:          return extended;
74:      }
75:
76:      public void setExtended(double extended)
77:      {
78:          this.extended = extended;
79:      }
80:
81:      @XmlAttribute
82:      public String getNumber()
83:      {
84:          return number;
85:      }
86:
87:      public void setNumber(String number)
88:      {
89:          this.number = number;
90:      }
91: }
```

```
1: package beans;
2:
3: import java.util.List;
4:
5: import javax.xml.bind.annotation.XmlElement;
6: import javax.xml.bind.annotation.XmlRootElement;
7:
8: @XmlRootElement
9: public class ItemsWrapper
10: {
11:     @XmlElement(name="item")
12:     private List<ItemBean> list;
13:
14:     public ItemsWrapper()
15:     {
16:         super();
17:     }
18:
19:     public List<ItemBean> getItemsList()
20:     {
21:         return list;
22:     }
23:
24:     public void setItemsList(List<ItemBean> list)
25:     {
26:         this.list = list;
27:     }
28: }
```

```
1: package beans;
2:
3: import java.util.Calendar;
4:
5: import javax.xml.bind.annotation.XmlAttribute;
6: import javax.xml.bind.annotation.XmlElement;
7: import javax.xml.bind.annotation.XmlRootElement;
8: import javax.xml.bind.annotation.XmlSchemaType;
9:
10: // Bean class for single purchase order
11: @XmlRootElement(name="order")
12: public class POBean
13: {
14:     // @XmlElement
15:     private CustomerBean customer;
16:     // @XmlElement(name="items")
17:     private ItemsWrapper items;
18:     // @XmlElement
19:     private double total;
20:     // @XmlElement
21:     private double shipping;
22:     // @XmlElement(name="HST")
23:     private double hst;
24:     // @XmlElement
25:     private double grandTotal;
26:     // @XmlAttribute
27:     private int id;
28:     // @XmlAttribute
29:     // @XmlSchemaType(name = "date")
30:     private Calendar submitted;
31:
32:     public POBean()
33:     {
34:         super();
35:     }
36:
37:     public POBean(CustomerBean customer, ItemsWrapper items, double total,
38:         double shipping, double hst, double grandTotal, int id,
39:         Calendar submitted)
40:     {
41:         this.customer = customer;
42:         this.items = items;
43:         this.total = total;
44:         this.shipping = shipping;
45:         this.hst = hst;
46:         this.grandTotal = grandTotal;
47:         this.id = id;
48:         this.submitted = submitted;
49:     }
50:
51:     // ===== getters and setters =====
52:     @XmlElement
53:     public CustomerBean getCustomer()
54:     {
55:         return customer;
56:     }
57:
58:     public void setCustomer(CustomerBean customer)
59:     {
60:         this.customer = customer;
61:     }
62:
63:     @XmlElement(name="items")
64:     public ItemsWrapper getItems()
65:     {
66:         return items;
67:     }
68:
69:     public void setItems(ItemsWrapper items)
70:     {
```

```

71:         this.items = items;
72:     }
73:
74:     @XmlElement
75:     public double getTotal()
76:     {
77:         return total;
78:     }
79:
80:     public void setTotal(double total)
81:     {
82:         this.total = total;
83:     }
84:
85:     @XmlElement
86:     public double getShipping()
87:     {
88:         return shipping;
89:     }
90:
91:     public void setShipping(double shipping)
92:     {
93:         this.shipping = shipping;
94:     }
95:
96:     @XmlElement
97:     public double getHst()
98:     {
99:         return hst;
100:    }
101:
102:    public void setHst(double hst)
103:    {
104:        this.hst = hst;
105:    }
106:
107:    @XmlElement
108:    public double getGrandTotal()
109:    {
110:        return grandTotal;
111:    }
112:
113:    public void setGrandTotal(double grandTotal)
114:    {
115:        this.grandTotal = grandTotal;
116:    }
117:
118:    @XmlAttribute
119:    public int getId()
120:    {
121:        return id;
122:    }
123:
124:    public void setId(int id)
125:    {
126:        this.id = id;
127:    }
128:
129:    @XmlAttribute
130:    @XmlSchemaType(name = "date")
131:    public Calendar getSubmitted()
132:    {
133:        return submitted;
134:    }
135:
136:    public void setSubmitted(Calendar submitted)
137:    {
138:        this.submitted = submitted;
139:    }
140: }

```

```
1: package beans;
2:
3: import javax.xml.bind.annotation.XmlElement;
4: import javax.xml.bind.annotation.XmlRootElement;
5:
6: @XmlRootElement
7: public class POFileBean
8: {
9:     @XmlElement
10:    private String url;
11:    @XmlElement
12:    private int id;
13:
14:    public POFileBean()
15:    {
16:        super();
17:    }
18:
19:    public String getUrl()
20:    {
21:        return url;
22:    }
23:
24:    public int getId()
25:    {
26:        return id;
27:    }
28: }
```

```
1: package beans;
2:
3: import java.util.List;
4:
5: import javax.xml.bind.annotation.XmlElement;
6: import javax.xml.bind.annotation.XmlRootElement;
7:
8: @XmlRootElement(name="PurchaseOrderFiles")
9: public class POFilesWrapper
10: {
11:     @XmlElement(name="PurchaseOrderFile")
12:     private List<POFileBean> lsPof;
13:
14:     public POFilesWrapper()
15:     {
16:         super();
17:     }
18:
19:     public List<POFileBean> getPOFileBeans()
20:     {
21:         return lsPof;
22:     }
23: }
```

```
1: package ctrl;
2:
3: import java.io.File;
4: import java.net.MalformedURLException;
5: import java.net.URL;
6: import java.text.SimpleDateFormat;
7: import java.util.Date;
8:
9: import b2b.Aggregator;
10: import b2b.Purchaser;
11: import b2b.Wholesaler;
12: import beans.AggOrderBean;
13:
14: public class Manager
15: {
16:     public static void main(String[] args)
17:     {
18:         // create aggregator and purchaser
19:         Aggregator agg = new Aggregator();
20:         Purchaser pch = new Purchaser();
21:
22:         // add wholesalers
23:         try
24:         {
25:             pch.addWholesaler(new Wholesaler("Toronto", new URL("http:
//roumani.eecs.yorku.ca:4413/axis/YYZ.jws")));
26:             pch.addWholesaler(new Wholesaler("Vancouver", new URL("htt
p://roumani.eecs.yorku.ca:4413/axis/YVR.jws")));
27:             pch.addWholesaler(new Wholesaler("Halifax", new URL("http:
//roumani.eecs.yorku.ca:4413/axis/YHZ.jws")));
28:         }
29:         catch (MalformedURLException m)
30:         {
31:             System.out.println("Invalid URL for wholesalers. Program a
borted.");
32:             System.exit(1);
33:         }
34:
35:         // set purchase key
36:         pch.setKey("4413secret");
37:
38:         // download PO files from eFood server
39:         PODownloader dl = new PODownloader("http://localhost:4413/eFoods/b
ackend/orders", "http://localhost:4413");
40:         Date now = new Date();
41:         SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd_HH-mm-ss")
;
42:         String dir = "." + sdf.format(now) + "/";
43:         new File(dir).mkdir();
44:
45:         try
46:         {
47:             String[] fns = dl.downloadPOs(dir);
48:
49:             int id = Math.abs(now.hashCode());
50:             dl.setStatus("pending");
51:             AggOrderBean prcrm = agg.getAggregatedOrder(fns, id);
52:             pch.purchase(prcrm, dir + "rpt" + id + ".xml");
53:             dl.setStatus("purchased");
54:             //dl.setStatus("new"); //restore for testing
55:         }
56:         catch (Exception e)
57:         {
58:             System.out.println(e.getMessage() + " Program aborted.");
59:             System.exit(1);
60:         }
61:     }
62:
63: }
```



```
1: package ctrl;
2:
3: import java.io.File;
4: import java.io.FileOutputStream;
5: import java.net.ConnectException;
6: import java.net.HttpURLConnection;
7: import java.net.MalformedURLException;
8: import java.net.URL;
9: import java.nio.channels.Channels;
10: import java.nio.channels.ReadableByteChannel;
11: import java.util.ArrayList;
12: import java.util.Iterator;
13: import java.util.List;
14:
15: import javax.xml.bind.JAXBContext;
16: import javax.xml.bind.Unmarshaller;
17:
18: import beans.POFileBean;
19: import beans.POFilesWrapper;
20:
21: public class PODownloader
22: {
23:     String pofwurl;
24:     String bseurl;
25:     List<POFileBean> pofs;
26:
27:     public PODownloader(String poFilesWrapperUrl, String baseUrl)
28:     {
29:         this.pofwurl = poFilesWrapperUrl;
30:         this.bseurl = baseUrl;
31:     }
32:
33:     public String[] downloadPOs(String dir) throws Exception
34:     {
35:         // download XML file that specifies urls of PO files
36:         String pofsname = dir + "POFiles.xml";
37:         this.download(pofwurl, pofsname);
38:
39:         // get list of PO file urls
40:         List<String> urls = getPOUrls(pofsname);
41:
42:         // download the PO files
43:         String[] fnames = new String[urls.size()];
44:         int i = 0;
45:         Iterator<String> itr = urls.iterator();
46:         while (itr.hasNext())
47:         {
48:             String cur = itr.next();
49:             fnames[i] = dir + this.getFileNameOnly(cur);
50:             this.download(cur, fnames[i]);
51:             i++;
52:         }
53:
54:         return fnames;
55:     }
56:
57:     private void download(String srcurl, String fname) throws Exception
58:     {
59:         try
60:         {
61:             URL url = new URL(srcurl);
62:             ReadableByteChannel src = Channels.newChannel(url.openStre
63:             am());
64:             FileOutputStream dst = new FileOutputStream(fname);
65:             dst.getChannel().transferFrom(src, 0, Long.MAX_VALUE);
66:             dst.close();
67:         }
68:         catch (MalformedURLException m)
69:         {
70:             throw new Exception("Invalid URL!");
71:         }
72:     }
73: }
```

```
70:         }
71:         catch (ConnectException c)
72:         {
73:             throw new Exception("Cannot connect to source URL.");
74:         }
75:         catch (Exception e)
76:         {
77:             //e.printStackTrace();
78:             throw new Exception("Error: fail to download file.");
79:         }
80:     }
81:
82:     private List<String> getPOUrls(String fname) throws Exception
83:     {
84:         if (fname == null)
85:             throw new NullPointerException("invalid file names.");
86:
87:         // define unmarshaller
88:         JAXBContext jc;
89:         Unmarshaller um;
90:         try
91:         {
92:             jc = JAXBContext.newInstance(POFilesWrapper.class);
93:             um = jc.createUnmarshaller();
94:         }
95:         catch (Exception e)
96:         {
97:             throw new Exception("Error occurs when defining unmarshall
er.");
98:         }
99:
100:        // unmarshal order files
101:        POFilesWrapper pofw;
102:        try
103:        {
104:            pofw = (POFilesWrapper) um.unmarshal(new File(fname));
105:        }
106:        catch (Exception e)
107:        {
108:            throw new Exception("Error occurs during unmarshalling.");
109:        }
110:
111:        if (pofw == null)
112:            throw new Exception("POFilesWrapper does not generated!");
113:
114:
115:        List<String> urls = new ArrayList<String>();
116:        this.pofs = pofw.getPOFileBeans();
117:        if (pofs != null) {
118:            Iterator<POFileBean> itr = pofs.iterator();
119:            while (itr.hasNext())
120:            {
121:                urls.add(this.bseurl + itr.next().getUrl());
122:            }
123:        }
124:
125:
126:        return urls;
127:    }
128:
129:    private String getFileNameOnly(String url)
130:    {
131:        return url.substring(url.lastIndexOf('/') + 1);
132:    }
133:
134:    public void setStatus(String status) throws Exception {
135:        if (pofs != null)
136:        {
137:            String type = "application/x-www-form-urlencoded";
138:            for(POFileBean pof: pofs) {
```

```

139:
140:
141:
142:
143:
144:
145:
146:
147:
148:
149:
150:
151:
152:
153:
154:
155:
156:
157:
158:
159:
160: }

int id = pof.getId();
try {
    URL u = new URL(this.bseurl + "/eFoods/bac
kend/orders/" + id + "?ajax=1&status=" + status);
    HttpURLConnection conn = (HttpURLConnection
n) u.openConnection();
    conn.setDoOutput(true);
    conn.setRequestMethod("POST");
    conn.setRequestProperty("Content-Type", t
ype );
    conn.setRequestProperty("Content-Length",
"0");
    System.out.println("State changed\t:" + id
+ ":" +
    status + ":" +
    conn.getResponseMessage())
;
    conn.disconnect();
} catch (Exception e) {
    //e.printStackTrace();
    //System.out.println(e.getMessage());
    throw new Exception("set status failed! "
+ e.getMessage());
}
} // end of traversal of POFileBean list
} // end of method setStatus

```

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <faceted-project>
3:   <runtime name="Apache Tomcat v8.0"/>
4:   <fixed facet="jst.web"/>
5:   <fixed facet="java"/>
6:   <fixed facet="wst.jsdt.web"/>
7:   <installed facet="java" version="1.7"/>
8:   <installed facet="jst.web" version="3.1"/>
9:   <installed facet="wst.jsdt.web" version="1.0"/>
10: </faceted-project>
```

```
1: package ctrl;
2:
3: import java.io.IOException;
4:
5: import javax.servlet.Servlet;
6: import javax.servlet.ServletException;
7: import javax.servlet.http.HttpServletRequest;
8: import javax.servlet.http.HttpServletResponse;
9:
10: import util.Route;
11:
12: public class Analytics extends BaseCtrl implements Servlet
13: {
14:     private static final long serialVersionUID = 1L;
15:     /*
16:      * Route definitions for this controller
17:      */
18:     public static final int ANALYTICS_PAGE = 0x0a;
19:
20:     @Override
21:     protected void processRequest(HttpServletRequest request,
22:         HttpServletResponse response) throws ServletException, IOException
23:     {
24:
25:         Route route = getRoute(request);
26:         switch(route.getIdentifier()) {
27:             case ANALYTICS_PAGE:
28:                 // get analytic information from application scope
29:                 Long avgAddItemTime = (Long) this.getServletContext().getA
30: ttribute("avgAddItemTime");
31:                 Long avgCheckOutTime = (Long) this.getServletContext().get
32: Attribute("avgCheckOutTime");
33:                 if (avgAddItemTime == null) avgAddItemTime = new Long(0);
34:                 if (avgCheckOutTime == null) avgCheckOutTime = new Long(0)
35: ;
36:                 // poke information into request scope
37:                 request.setAttribute("avgAddItemTime", avgAddItemTime);
38:                 request.setAttribute("avgCheckOutTime", avgCheckOutTime);
39:                 // dispatch to JSPX
40:                 request.getRequestDispatcher("/partials/_analytics.jspx").
41: forward(request, response);
42:                 break;
43:             default:
44:                 throw new ServletException("Route configuration error.");
45:         }
46:     }
47: }
```

```

1: package ctrl;
2:
3: import javax.servlet.Servlet;
4: import javax.servlet.ServletException;
5: import javax.servlet.http.HttpServletRequest;
6: import javax.servlet.http.HttpServletResponse;
7:
8: import model.UserBean;
9: import util.Route;
10: import util.SSOAuthenticator;
11:
12: /**
13:  * Servlet implementation class Login
14:  */
15: public class Auth extends BaseCtrl implements Servlet {
16:     private static final long serialVersionUID = 1L;
17:
18:     /*
19:      * Route definitions for this controller
20:      */
21:     public static final int ROUTE_INITIAL = 0x0a;
22:     public static final int ROUTE_AUTHENTICATE = 0x0b;
23:     public static final int ROUTE_LOGOUT = 0x0c;
24:     public static final int ROUTE_USER_BADGE = 0x0d;
25:
26:
27:     protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws Exception {
28:         SSOAuthenticator auth = (SSOAuthenticator) getAuthenticator(request);
29:         Route route = getRoute(request);
30:         switch(route.getIdentifier()) {
31:             case ROUTE_INITIAL:
32:                 request.getSession().setAttribute("LAST_STATE", request.getParameter("state"));
33:                 auth.SSORedirect(request, response);
34:                 break;
35:             case ROUTE_AUTHENTICATE:
36:                 if (auth.login(request, null, null)) {
37:                     UserBean usr = auth.getUser(request);
38:                     System.out.println("Login: Successfully authenticated as " + usr.getName());
39:                     String state = ((String) request.getSession().getAttribute("LAST_STATE"));
40:                     if (state == null || state.startsWith("backend/error")) { state = ""; }
41:                     String rdr = (request.getContextPath().length() == 0 ? "/" : request.getContextPath()) + "#!" + state;
42:                     System.out.println("Redirecting to: " + rdr);
43:                     response.sendRedirect(rdr);
44:                 } else {
45:                     throw new Exception("Sorry! We could not authenticate you at this moment.");
46:                 }
47:                 break;
48:             case ROUTE_LOGOUT:
49:                 auth.logout(request);
50:                 System.out.println("Login: Logged out");
51:                 break;
52:             case ROUTE_USER_BADGE:
53:                 request.setAttribute("currentUser", auth.getUser(request));
54:                 request.getRequestDispatcher("/partials/_userBadge.jsp").forward(request, response);
55:                 break;
56:             default:
57:                 throw new ServletException("Route configuration error.");
58:         }
59:     }
60:

```

```
61:         }  
62:  
63: }
```

```

1: package ctrl;
2: import java.io.IOException;
3:
4: import javax.servlet.ServletException;
5: import javax.servlet.http.HttpServlet;
6: import javax.servlet.http.HttpServletRequest;
7: import javax.servlet.http.HttpServletResponse;
8:
9: import util.Authenticator;
10: import util.Route;
11: import util.Router;
12: import model.*;
13:
14:
15: public abstract class BaseCtrl extends HttpServlet
16: {
17:     private static final long serialVersionUID = 1L;
18:
19:     public class PagingHelper {
20:         int page;
21:         int limit;
22:         /**
23:          * @return the page
24:          */
25:         public int getPage() {
26:             return page;
27:         }
28:
29:         /**
30:          * @return the limit
31:          */
32:         public int getLimit() {
33:             return limit;
34:         }
35:
36:         public PagingHelper(int page, int limit) {
37:             super();
38:             this.page = page;
39:             this.limit = limit;
40:         }
41:     }
42:
43:     public BaseCtrl() {
44:         super();
45:
46:         // TODO Auto-generated constructor stub
47:     }
48:
49:     /**
50:      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
response)
51:      */
52:     protected void doGet(HttpServletRequest request, HttpServletResponse respo
nse) throws ServletException, IOException {
53:         try {
54:             processRequest(request, response);
55:         } catch (Exception e) {
56:             e.printStackTrace();
57:             throw new ServletException(e.getMessage());
58:         }
59:
60:     }
61:
62:
63:     /**
64:      * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
65:      */
66:     protected void doPost(HttpServletRequest request, HttpServletResponse resp
onse) throws ServletException, IOException {

```



```

67:         doGet(request, response);
68:     }
69:
70:     protected abstract void processRequest(HttpServletRequest request, HttpSer
vletResponse response) throws Exception;
71:
72:
73:     protected PagingHelper getPagination(HttpServletRequest request) {
74:         String pageStr, limitStr;
75:         int page, limit;
76:         if ((pageStr = request.getParameter("page")) != null )
77:             page = Integer.parseInt(pageStr);
78:         else
79:             page = BaseDAO.PAGE_ALL;
80:
81:         if ((limitStr = request.getParameter("limit")) != null)
82:             limit = Integer.parseInt(limitStr);
83:         else
84:             limit = BaseDAO.LIMIT_ALL;
85:         return new PagingHelper(page, limit);
86:     }
87:
88:     //Static helpers
89:
90:     protected static EFoods getModel(HttpServletRequest request) {
91:         return (EFoods) request.getServletContext().getAttribute(ctrl.Fron
t.MODEL_KEY);
92:     }
93:
94:     protected static Route getRoute(HttpServletRequest request) {
95:         return (Route) request.getAttribute(Router.REQUEST_ROUTE_KEY);
96:     }
97:
98:     protected static Authenticator getAuthenticator(HttpServletRequest request
) {
99:         return (Authenticator) request.getServletContext().getAttribute(ct
rl.Front.SSO_AUTHENTICATOR_KEY);
100:     }
101:
102: }

```

```

1: package ctrl;
2:
3: import java.util.ArrayList;
4: import java.util.List;
5:
6: import javax.servlet.Servlet;
7: import javax.servlet.ServletException;
8: import javax.servlet.http.HttpServletRequest;
9: import javax.servlet.http.HttpServletResponse;
10:
11: import util.PurchaseOrderFiles;
12: import util.Route;
13: import util.SSOAuthenticator;
14: import util.PurchaseOrderFiles.PurchaseOrderFile;
15: import model.CartModel;
16: import model.EFoods;
17: import model.ItemBean;
18: import model.ItemDAO;
19:
20: /**
21:  * Servlet implementation class Cart
22:  */
23:
24: public class Cart extends BaseCtrl implements Servlet {
25:     private static final long serialVersionUID = 1L;
26:     /*
27:      * Route definitions for this controller
28:      */
29:     public static final int ROUTE_ALL = 0x0a;
30:     public static final int ROUTE_MANIPULATE = 0x0b;
31:     public static final int ROUTE_HISTORY = 0x0c;
32:     public static final int ROUTE_CHECKOUT = 0x0d;
33:     public static final int ROUTE_BADGE = 0x0e;
34:     public static final int ROUTE_SUCCESS = 0x0f;
35:
36:     /**
37:      * @see BaseCtrl#BaseCtrl()
38:      */
39:     public Cart() {
40:         super();
41:         // TODO Auto-generated constructor stub
42:     }
43:
44:
45:     @Override
46:     protected void processRequest(HttpServletRequest request,
47:         HttpServletResponse response) throws Exception {
48:         Route route = getRoute(request);
49:         SSOAuthenticator auth = (SSOAuthenticator) getAuthenticator(request);
50:         CartModel cart = CartModel.getInstance(request, auth);
51:         EFoods model = getModel(request);
52:         PagingHelper paging = getPagination(request);
53:         String filePath = request.getServletContext().getRealPath(Orders.B
ASE_PATH);
54:         switch(route.getIdentifier()) {
55:             case ROUTE_ALL:
56:                 request.setAttribute("cartItems", cart.getCartItems());
57:                 request.setAttribute("total", cart.getTotal());
58:                 request.setAttribute("shipping", cart.getShipping());
59:                 request.setAttribute("tax", cart.getTax());
60:                 request.setAttribute("user", auth.getUser(request));
61:                 request.getRequestDispatcher("/partials/_cart.jsp").forwa
rd(
62:                     request, response);
63:                 break;
64:             case ROUTE_MANIPULATE:
65:                 String itemNumber;
66:                 ItemBean item;
67:                 int newQty;

```

```

68:         itemNumber = request.getParameter("item");
69:         try {
70:             newQty = Integer.parseInt(request.getParameter("q
ty"));
71:         } catch (NumberFormatException e) {
72:             throw new Exception("Quantity must be a valid numb
er.");
73:         }
74:
75:         if (newQty < 0) throw new Exception("Quantity must be 0 or
positive.");
76:
77:         item = model.items(itemNumber, ItemDAO.CAT_ALL, paging.get
Page(), paging.getLimit(), ItemDAO.NO_FILTER).get(0);
78:         cart.manipulateCart(item, newQty);
79:
80:         // poke number of cart items into session scope, for liste
ner
81:         request.getSession().setAttribute("cartItemsCount", cart.g
etCartItems().size());
82:
83:         break;
84:     case ROUTE_CHECKOUT:
85:         System.out.println("Checking out");
86:         if (cart.getCartItems().size() < 1) {
87:             throw new Exception("Can't checkout empty cart!");
88:         }
89:         int orderId = model.createPurchaseOrder(cart, filePath);
90:         // notify listener by poking attribute into session scope
after client had checked out
91:         cart.clear();
92:         request.getSession().setAttribute("checkedOut", true);
93:         String rdr = (request.getContextPath().length() == 0 ? "/"
: request.getContextPath()) + "##!backend/cart/success?order=" + orderId;
94:         response.sendRedirect(rdr);
95:         break;
96:     case ROUTE_HISTORY:
97:         PurchaseOrderFiles pofs = new PurchaseOrderFiles(filePath)
;
98:         List<PurchaseOrderFile> pofList = pofs.getOrdersByUser(auth
h.getUser(request).getName());
99:
100:         List<PurchaseOrderFile> pendingOrders = new ArrayList<Purc
haseOrderFile>();
101:         List<PurchaseOrderFile> newOrders = new ArrayList<Purchase
OrderFile>();
102:         List<PurchaseOrderFile> purchasedOrders = new ArrayList<Pu
rchaseOrderFile>();
103:
104:         //Sorting the purchase orders by their type
105:         for(PurchaseOrderFile pof : pofList)
106:             if (pof.getStatus().equals(PurchaseOrderFile.STATU
S_NEW))
107:                 newOrders.add(pof);
108:             else if (pof.getStatus().equals(PurchaseOrderFile.
STATUS_PENDING))
109:                 pendingOrders.add(pof);
110:             else if (pof.getStatus().equals(PurchaseOrderFile.
STATUS_PURCHASED))
111:                 purchasedOrders.add(pof);
112:
113:         request.setAttribute("pendingOrders", pendingOrders);
114:         request.setAttribute("newOrders", newOrders);
115:         request.setAttribute("purchasedOrders", purchasedOrders);
116:         request.getRequestDispatcher("/partials/_cartHistory.jspx"
).forward(
117:             request, response);
118:         break;
119:     case ROUTE_SUCCESS:
120:         request.getRequestDispatcher("/partials/_cartSuccess.jspx"

```

```

).forward(
    121:                                     request, response);
    122:                                     break;
    123:     case ROUTE_BADGE:
    124:         request.setAttribute("cartItemCount", cart.getCartItems().
size());
    125:         request.getRequestDispatcher("/partials/_cartBadge.jspx").
forward(
    126:                                     request, response);
    127:                                     break;
    128:     default:
    129:         throw new ServletException("Route configuration error.");
    130:     }
    131:
    132: }
    133:
    134:
    135:
    136:
    137:
    138:
    139:
    140: }
```

```

1: package ctrl;
2:
3: import java.util.List;
4:
5: import javax.servlet.ServletException;
6: import javax.servlet.annotation.WebServlet;
7: import javax.servlet.http.HttpServletRequest;
8: import javax.servlet.http.HttpServletResponse;
9:
10: import util.Route;
11: import model.*;
12:
13: @WebServlet(name = "Category", displayName = "Category")
14: public class Category extends BaseCtrl {
15:     private static final long serialVersionUID = 1L;
16:
17:     /*
18:      * Route definitions for this controller
19:      */
20:     public static final int ROUTE_ALL = 0xff;
21:     public static final int ROUTE_BY_ID = 0x0a;
22:
23:     protected void processRequest(HttpServletRequest request,
24:                                     HttpServletResponse response) throws Exception {
25:         EFoods model = getModel(request);
26:         Route route = getRoute(request);
27:         List<CategoryBean> results = null;
28:         int routeType;
29:         if (route == null)
30:             routeType = -1;
31:         else
32:             routeType = route.getIdentifier();
33:
34:         switch (routeType) {
35:             case ROUTE_BY_ID:
36:                 int id = Integer.parseInt(route.getMatcher().group("catId"
37: ));
38:                 results = model.categories(id);
39:                 break;
40:             case ROUTE_ALL:
41:                 results = model.categories(CategoryDAO.ID_ALL);
42:                 break;
43:             default:
44:                 throw new ServletException("Route configuration error.");
45:         }
46:         request.setAttribute("results", results);
47:         request.getRequestDispatcher("/partials/_category.jsp").forward(
48:             request, response);
49:
50:     }
51:
52: }

```

```

1: package ctrl;
2:
3: import java.io.IOException;
4: import java.util.ArrayList;
5: import java.util.List;
6:
7: import javax.servlet.ServletConfig;
8: import javax.servlet.ServletException;
9: import javax.servlet.annotation.WebServlet;
10: import javax.servlet.http.HttpServlet;
11: import javax.servlet.http.HttpServletRequest;
12: import javax.servlet.http.HttpServletResponse;
13:
14: import org.apache.tomcat.util.codec.binary.Base64;
15:
16: import model.*;
17: import util.*;
18:
19: /**
20:  * Servlet implementation class Front
21:  */
22: @WebServlet("/backend/*")
23: @SuppressWarnings("serial")
24: public class Front extends HttpServlet {
25:     private static final long serialVersionUID = 1L;
26:
27:     //Context objects
28:     public static final String MODEL_KEY = "com.eFoods.ctrl.Front.EFOODS";
29:     private static final String ROUTER_KEY = "com.eFoods.ctrl.Front.ROUTER";
30:     public static final String SSO_AUTHENTICATOR_KEY = "com.eFoods.ctrl.Front.
HTTP_AUTHENTICATOR";
31:
32:
33:     //SSO Configuration
34:     //private static final String SSO_AUTHENTICATOR_URL = "http://localhost:90
00/";
35:     private static final String SSO_AUTHENTICATOR_URL = "https://www.cse.yorku
.ca/~cse11011/4413/auth/";
36:     private static final String SSO_RECIEVER_URL = "backend/login/authenticate
";
37:     private static final String SSO_SHARED_KEY = "7fcbc0e27e7bb31e4ad7a39677ee
bdaf6b94c9db7dcdb2ebf25791298609a4a4";
38:     private static final List<String> SSO_ADMINS = new ArrayList<String>() {{
39:         add("cse11011"); //List of admin users
40:         add("cse13195");
41:         add("cse13122");
42:     }};
43:
44:
45: /**
46:  * @see HttpServlet#HttpServlet()
47:  */
48: public Front() {
49:     super();
50:     // TODO Auto-generated constructor stub
51: }
52:
53: /**
54:  * @see Servlet#init(ServletConfig)
55:  */
56: public void init(ServletConfig config) throws ServletException {
57:     super.init();
58:
59:     //Initializing Model
60:     EFoods model = new EFoods();
61:
62:     //Initializing authentication module
63:     SSOAuthenticator auth;
64:     try {
65:         auth = new SSOAuthenticator(SSO_AUTHENTICATOR_URL, SSO_REC

```

```

IEVER_URL, SSO_SHARED_KEY, SSO_ADMINS);
66:         } catch (Exception e) {
67:             throw new ServletException(e.getMessage());
68:         }
69:
70:
71:         Router appRouter = new Router();
72:         /*
73:          * Start Route Definitions
74:          */
75:         //List All categories
76:         appRouter.addRoute(new Route("^/category(/)?$", "Category", Route.M
ETHOD_GET, Category.ROUTE_ALL, false));
77:         //List All items from a category
78:         appRouter.addRoute(new Route("^/category/(?<catId>[0-9]+)(/)?$", "I
tem", Route.METHOD_GET, Item.ROUTE_BY_CATEGORY, false));
79:         //List all items
80:         appRouter.addRoute(new Route("^/item(/)?$", "Item", Route.METHOD_GE
T, Item.ROUTE_ALL, false));
81:         //Render a partial of a single item
82:         appRouter.addRoute(new Route("^/item/(?<itemNumber>[0-9a-zA-Z]+)/p
artial(/)?$", "Item", Route.METHOD_GET, Item.ROUTE_BY_NUMBER, false));
83:         //Render an item as a result listing
84:         appRouter.addRoute(new Route("^/item/(?<itemNumber>[0-9a-zA-Z]+)(/
)?$", "Item", Route.METHOD_GET, Item.ROUTE_BY_NUMBER_FULL, false));
85:         //Free text search on all items
86:         appRouter.addRoute(new Route("^/search(/)?$", "Item", Route.METHOD_
GET, Item.ROUTE_BY_SEARCH, false));
87:         //Authentication initiation endpoint
88:         appRouter.addRoute(new Route("^/login(/)?$", "Auth", Route.METHOD_G
ET, Auth.ROUTE_INITIAL, false));
89:         //Authentication redirect reciever endpoint
90:         appRouter.addRoute(new Route("^/login/authenticate(/)?$", "Auth", R
oute.METHOD_GET, Auth.ROUTE_AUTHENTICATE, false));
91:         //Log out current user
92:         appRouter.addRoute(new Route("^/logout(/)?$", "Auth", Route.METHOD_
GET, Auth.ROUTE_LOGOUT, true));
93:         //Render the badge that shows user information or logged in link i
f not auth
94:         appRouter.addRoute(new Route("^/user(/)?$", "Auth", Route.METHOD_GE
T, Auth.ROUTE_USER_BADGE, false));
95:         //Render current cart
96:         appRouter.addRoute(new Route("^/cart(/)?$", "Cart", Route.METHOD_GE
T, Cart.ROUTE_ALL, false));
97:         //Change elements in the cart. Accepted GET "item=?&qty=?
98:         appRouter.addRoute(new Route("^/cart(/)?$", "Cart", Route.METHOD_PO
ST, Cart.ROUTE_MANIPULATE, false));
99:         //Initiate checkout of the current cart
100:         appRouter.addRoute(new Route("^/cart/checkout(/)?$", "Cart", Route.
METHOD_POST, Cart.ROUTE_CHECKOUT, true));
101:         //Render purchase history page
102:         appRouter.addRoute(new Route("^/cart/history(/)?$", "Cart", Route.M
ETHOD_GET, Cart.ROUTE_HISTORY, true));
103:         //Render checkout success page
104:         appRouter.addRoute(new Route("^/cart/success(/)?$", "Cart", Route.M
ETHOD_GET, Cart.ROUTE_SUCCESS, true));
105:         //Render cart item count badge
106:         appRouter.addRoute(new Route("^/cart/badge(/)?$", "Cart", Route.MET
HOD_GET, Cart.ROUTE_BADGE, false));
107:         //Render generic error page (accepts b64 string as an error messag
e to show)
108:         appRouter.addRoute(new Route("^/error/(?<Base64EncodedMessage>.*)?
", "Misc", Route.METHOD_GET, Misc.ERROR_PAGE, false));
109:         //Render analytics page
110:         appRouter.addRoute(new Route("^/analytics(/)?$", "Analytics", Route
.METHOD_ANY, Analytics.ANALYTICS_PAGE, true, true));
111:         //Render XML listing of available orders
112:         appRouter.addRoute(new Route("^/orders(/)?$", "Orders", Route.METHO
D_GET, Orders.ROUTE_ALL_NEW, false));
113:         //Change status of an order (Accepts GET parameter "state" being o

```

```

ne of "new", "pending" or "purchased")
114:         appRouter.addRoute(new Route("^/orders/(?<orderId>[0-9]+)(/)?$","O
rders", Route.METHOD_POST, Orders.ROUTE_UPDATE_STATUS, false));
115:         //Permenantly remove all orders from the system. Destructive actio
n!
116:         appRouter.addRoute(new Route("^/orders/nuke(/)?$","Orders", Route.
METHOD_GET, Orders.ROUTE_NUKE, false));
117:         /*
118:         * End Route definitions
119:         */
120:
121:         //Poking the context
122:         config.getServletContext().setAttribute(MODEL_KEY, model);
123:         config.getServletContext().setAttribute(ROUTER_KEY, appRouter);
124:         config.getServletContext().setAttribute(SSO_AUTHENTICATOR_KEY, aut
h);
125:
126:     }
127:
128: /**
129:  * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
response)
130:  */
131:     protected void doGet(HttpServletRequest request, HttpServletResponse respo
nse) throws ServletException, IOException {
132:
133:         // ensure to create a session whenever a fresh request comes
134:         request.getSession();
135:
136:         Router router = (Router) request.getServletContext().getAttribute
(ROUTER_KEY);
137:         Authenticator httpAuth = (SSOAuthenticator) request.getServletContext
().getAttribute(SSO_AUTHENTICATOR_KEY);
138:
139:         Route route;
140:         try {
141:             if ((route = router.getRoute(request)) != null) {
142:                 if (!route.isRequireAuthentication() || httpAuth.i
sAuthenticated(request)) {
143:                     System.out.println("Serving Route: " + rou
te.getMethod() + " " + request.getPathInfo() + " (" + route.getUriPattern() + ") " +
(route.isRequireAuthentica
tion() ? " (Authenticated as " + ((SSOAuthenticator) httpAuth).getUser(request).getName()
+ ")": ""));
145:                     request.getServletContext().getNamedDispat
cher(route.getDestination()).forward(request, response);
146:                 } else {
147:                     throw new Exception("You are not authorize
d to access this page. Please login.");
148:                 }
149:             } else {
150:                 System.out.println("Route not found!: " + request.
getPathInfo());
151:                 throw new Exception("Requested path is not found."
);
152:             }
153:         } catch (Exception e) {
154:             String exceptionMsg = e.getMessage();
155:             if (exceptionMsg == null) {
156:                 exceptionMsg = "Unknown error.";
157:             }
158:             System.out.println("Front controller exception! : " + exce
ptionMsg);
159:             String encodedError = Base64.encodeBase64String(exceptionM
sg.getBytes());
160:             if(request.getParameter("main") != null) {
161:                 System.out.println("Routing via Misc.");
162:                 request.setAttribute("encodedError", encodedError)
;
163:                 request.getServletContext().getNamedDispatcher("Mi

```



```
sc").forward(request, response);
164:                                     } else if (request.getParameter("ajax") != null) {
165:                                     response.setStatus(HttpServletResponse.SC_BAD_REQ
UEST);
166:                                     response.setContentType("application/json");
167:                                     response.getOutputStream().print("{\"error\": \""
+ exceptionMsg.replaceAll("\\\"", "\\\"\\\"") + "\"}");
168:                                     } else {
169:                                     String rdr = (request.getContextPath().length() ==
0 ? "/" : request.getContextPath()) + "##!backend/error/" + encodedError;;
170:                                     System.out.println("Redirecting to: " + rdr);
171:                                     response.sendRedirect(rdr);
172:                                     }
173:                                     }
174:                                     }
175:
176:                                     /**
177:                                     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
178:                                     */
179:                                     protected void doPost(HttpServletRequest request, HttpServletResponse resp
onse) throws ServletException, IOException {
180:                                     doGet(request, response);
181:                                     }
182:
183: }
```

```

1: package ctrl;
2:
3: import java.util.List;
4:
5:
6:
7: import javax.servlet.annotation.WebServlet;
8: import javax.servlet.http.HttpServletRequest;
9: import javax.servlet.http.HttpServletResponse;
10:
11: import util.Route;
12: import util.SSOAuthenticator;
13: import model.*;
14:
15:
16: @WebServlet(name = "Item", displayName = "Item")
17: public class Item extends BaseCtrl {
18:     private static final long serialVersionUID = 1L;
19:
20:     /*
21:      * Route definitions for this controller
22:      */
23:     public static final int ROUTE_ALL = 0xff;
24:     public static final int ROUTE_BY_NUMBER = 0x0b;
25:     public static final int ROUTE_BY_CATEGORY = 0x0c;
26:     public static final int ROUTE_BY_SEARCH = 0x0d;
27:     public static final int ROUTE_BY_NUMBER_FULL = 0x0e;
28:
29:     /**
30:      * Process the request and response. Uses the model, authenticator, route
31:      * to manage the request appropriately.
32:      */
33:     protected void processRequest(HttpServletRequest request,
34:                                     HttpServletResponse response) throws Exception {
35:         EFoods model = getModel(request);
36:         Route route = getRoute(request);
37:         PagingHelper paging = getPagination(request);
38:         SSOAuthenticator auth = (SSOAuthenticator) getAuthenticator(request);
39:
40:         CartModel cart = CartModel.getInstance(request, auth);
41:         List<ItemBean> results = null;
42:         int routeType;
43:         if (route == null)
44:             routeType = -1;
45:         else
46:             routeType = route.getIdentifier();
47:
48:         switch (routeType) {
49:             case ROUTE_BY_CATEGORY:
50:                 int catId = Integer.parseInt(route.getMatcher().group("categoryId"));
51:                 CategoryBean cat = model.categories(catId).get(0);
52:
53:                 results = model.items(ItemDAO.NUMBER_ALL, catId, paging.getPage(), paging.getLimit(), ItemDAO.NO_FILTER);
54:                 request.setAttribute("cartItems", cart.mappedCartItems());
55:                 request.setAttribute("category", cat);
56:                 request.setAttribute("results", results);
57:                 request.getRequestDispatcher("/partials/_items.jsp").forward(request,
58:                                     response);
59:                 break;
60:             case ROUTE_BY_NUMBER_FULL:
61:                 String number = route.getMatcher().group("itemNumber");
62:                 results = model.items(number, ItemDAO.CAT_ALL, ItemDAO.PAGE_ALL, ItemDAO.LIMIT_ALL, ItemDAO.NO_FILTER);
63:                 request.setAttribute("cartItems", cart.mappedCartItems());
64:                 request.setAttribute("category", results.get(0).getCategory());

```

```
65:         request.setAttribute("results", results);
66:         request.getRequestDispatcher("/partials/_items.jspx").forw
ard(request,
67:         response);
68:         break;
69:         case ROUTE_BY_NUMBER:
70:             String number2 = route.getMatcher().group("itemNumber");
71:             ItemBean item = model.items(number2, ItemDAO.CAT_ALL, Item
DAO.PAGE_ALL, ItemDAO.LIMIT_ALL, ItemDAO.NO_FILTER).get(0);
72:             request.setAttribute("cartItem", cart.getCartItemFor(item)
);
73:             request.setAttribute("item", item);
74:             request.getRequestDispatcher("/partials/_item.jspx").forwa
rd(request,
75:             response);
76:             break;
77:             case ROUTE_ALL:
78:                 results = model.items(ItemDAO.NUMBER_ALL, ItemDAO.CAT_ALL,
paging.getPage(), paging.getLimit(), ItemDAO.NO_FILTER);
79:                 request.setAttribute("results", results);
80:                 request.getRequestDispatcher("/partials/_items.jspx").forw
ard(request,
81:                 response);
82:                 break;
83:                 case ROUTE_BY_SEARCH:
84:                     String filter = request.getParameter("filter");
85:                     System.out.println(filter);
86:                     results = model.items(ItemDAO.NUMBER_ALL, ItemDAO.CAT_ALL,
paging.getPage(), paging.getLimit(), filter);
87:                     request.setAttribute("results", results);
88:                     request.getRequestDispatcher("/partials/_items.jspx").forw
ard(request,
89:                     response);
90:                     break;
91:                     default:
92:                         throw new Exception("routing error! not suppose to be here
");
93:                 }
94:             }
95:
96:
97:
98: }
```

```

1: package ctrl;
2:
3: import java.io.IOException;
4:
5: import javax.servlet.Servlet;
6: import javax.servlet.ServletException;
7: import javax.servlet.http.HttpServletRequest;
8: import javax.servlet.http.HttpServletResponse;
9:
10: import org.apache.tomcat.util.codec.binary.Base64;
11:
12: import util.Route;
13:
14: /**
15:  * Servlet implementation class Cart
16:  */
17:
18: public class Misc extends BaseCtrl implements Servlet {
19:     private static final long serialVersionUID = 1L;
20:     public static final int ERROR_PAGE = 0x0a;
21:     /**
22:      * @see BaseCtrl#BaseCtrl()
23:      */
24:     public Misc() {
25:         super();
26:         // TODO Auto-generated constructor stub
27:     }
28:
29:
30:     @Override
31:     protected void processRequest(HttpServletRequest request,
32:                                     HttpServletResponse response) throws ServletException, IOE
xception {
33:         Route route = getRoute(request);
34:
35:         int identifier = ERROR_PAGE;
36:
37:         switch(identifier) {
38:             case ERROR_PAGE:
39:                 String errorMessage;
40:                 String encodedError = (String) request.getAttribute("encod
edError");
41:                 if (encodedError == null) {
42:                     try {
43:                         encodedError = route.getMatcher().group("B
ase64EncodedMessage");
44:                         errorMessage= new String(Base64.decodeBase
64(encodedError));
45:                     } catch (Exception e) {
46:                         errorMessage = "Something went wrong somew
here! :(";
47:                     }
48:                 } else {
49:                     errorMessage= new String(Base64.decodeBase64(encod
edError));
50:                 }
51:
52:                 request.setAttribute("errorMessage", errorMessage);
53:                 request.getRequestDispatcher("/partials/_serverError.jspx"
).forward(
54:                     request, response);
55:                 break;
56:             default:
57:                 throw new ServletException("Routing configuration error.")
;
58:         }
59:
60:     }
61:
62:

```

```
63:
64:
65:
66:
67:
68: }
```

```

1: package ctrl;
2:
3: import java.io.StringWriter;
4: import java.util.ArrayList;
5: import java.util.List;
6:
7: import javax.servlet.ServletException;
8: import javax.servlet.http.HttpServletRequest;
9: import javax.servlet.http.HttpServletResponse;
10: import javax.xml.bind.JAXBContext;
11: import javax.xml.bind.Marshaller;
12: import javax.xml.transform.stream.StreamResult;
13:
14: import model.POFileWrapper;
15: import model.POFilesWrapper;
16: import util.PurchaseOrderFiles;
17: import util.PurchaseOrderFiles.PurchaseOrderFile;
18: import util.Route;
19:
20: public class Orders extends BaseCtrl {
21:
22:     /**
23:      *
24:      */
25:     private static final long serialVersionUID = 1L;
26:
27:     public static final int ROUTE_ALL_NEW = 0x0a;
28:     public static final int ROUTE_UPDATE_STATUS = 0x0b;
29:     public static final int ROUTE_NUKE = 0x0c;
30:     public static final String BASE_PATH = "/PurchaseOrders/";
31:
32:
33:
34:     @Override
35:     protected synchronized void processRequest(HttpServletRequest request,
36:         HttpServletResponse response) throws Exception {
37:         Route route = getRoute(request);
38:         String basePath = request.getServletContext().getRealPath(BASE_PATH
H);
39:         String baseUrl = request.getContextPath() + BASE_PATH;
40:         PurchaseOrderFiles pofs;
41:
42:         switch(route.getIdentifier()) {
43:             case ROUTE_ALL_NEW:
44:                 pofs = new PurchaseOrderFiles(basePath);
45:                 List<PurchaseOrderFile> pofList = pofs.getOrdersByStatus(P
urchaseOrderFile.STATUS_NEW);
46:                 List<POFileWrapper> pofWrapperList = new ArrayList<POFileW
rapper>();
47:                 for(PurchaseOrderFile pof: pofList) {
48:                     pofWrapperList.add(new POFileWrapper(baseUrl + pof
.getFileOnly(),
49:                         pof.getOrderID()));
50:                 }
51:                 POFilesWrapper pofsWrapper = new POFilesWrapper(pofWrapper
List);
52:                 String xmlOut = getXML(pofsWrapper);
53:                 response.getWriter().write(xmlOut);
54:
55:                 break;
56:             case ROUTE_UPDATE_STATUS:
57:                 int orderId = Integer.parseInt(route.getMatcher().group("o
rderId"));
58:                 String status = request.getParameter("status");
59:                 pofs = new PurchaseOrderFiles(basePath);
60:                 pofs.updateStatus(orderId, status);
61:                 break;
62:             case ROUTE_NUKE:
63:                 pofs = new PurchaseOrderFiles(basePath);
64:                 pofs.nuke();

```

```
65:         break;
66:     default:
67:         throw new ServletException("Routing configuration error.")
;
68:     }
69:
70: }
71:
72: private String getXML(Object wrapper) throws Exception {
73:     StringWriter sWriter = null;
74:     try {
75:         JAXBContext jaxbCtx = JAXBContext.newInstance(wrapper
76:             .getClass());
77:         Marshaller marshaller = jaxbCtx.createMarshaller();
78:
79:         marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
80:             Boolean.TRUE);
81:         sWriter = new StringWriter();
82:
83:         marshaller.marshal(wrapper, new StreamResult(sWriter));
84:
85:         // add the stylesheet
86:         StringBuffer sb = new StringBuffer(sWriter.toString());
87:
88:
89:         return sb.toString();
90:     } catch (Exception e) {
91:         throw e;
92:     } finally {
93:         if (sWriter != null)
94:             sWriter.close();
95:     }
96:
97:
98: }
99:
100: }
```

```

1: package filter;
2:
3: import java.io.IOException;
4: import java.util.HashMap;
5: import java.util.Map;
6:
7: import javax.servlet.DispatcherType;
8: import javax.servlet.Filter;
9: import javax.servlet.FilterChain;
10: import javax.servlet.FilterConfig;
11: import javax.servlet.RequestDispatcher;
12: import javax.servlet.ServletException;
13: import javax.servlet.http.HttpServletRequest;
14: import javax.servlet.http.HttpServletResponse;
15: import javax.servlet.ServletRequest;
16: import javax.servlet.ServletResponse;
17: import javax.servlet.annotation.WebFilter;
18:
19: import model.CartModel.CartItem;
20: import model.EFoods;
21: import model.ItemBean;
22: import model.ItemDAO;
23:
24: /**
25:  * Servlet Filter implementation class CrossSell
26:  */
27: @WebFilter(dispatcherTypes = { DispatcherType.REQUEST, DispatcherType.FORWARD, DispatcherType.INCLUDE }, urlPatterns = { "/backend/cart", "/partials/_item.jsp" })
28: @SuppressWarnings("unchecked")
29: public class CrossSell implements Filter {
30:
31:     @SuppressWarnings("serial")
32:     private static Map<String, String> TO_CROSS_SELL = new HashMap<String, String>() {
33:
34:         {
35:             put("0905A044", "0905A123"); //Add cross sell items <ItemAddedToCart,ItemToCrossSell>
36:             put("0905A112", "0905A127");
37:             put("0905A123", "0905A112");
38:             put("0905A127", "0905A044");
39:             put("1409S413", "2002H712");
40:         }
41:     };
42:
43:     public static String ACTIVE_CROSS_SELLS_MAP_KEY = "com.eFoods.filter.ACSMap";
44:
45:     /**
46:      * Default constructor.
47:      */
48:     public CrossSell() {
49:         // TODO Auto-generated constructor stub
50:     }
51:
52:     /**
53:      * @see Filter#destroy()
54:      */
55:     public void destroy() {
56:         // TODO Auto-generated method stub
57:     }
58:
59:     /**
60:      * @see Filter#doFilter(ServletRequest, ServletResponse, FilterChain)
61:      */
62:     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
63:         // TODO Auto-generated method stub
64:         // place your code here
65:         try {
66:             HttpServletRequest httpRequest = (HttpServletRequest) request;

```



```

est;
67:             HttpServletResponse httpResponse = (HttpServletResponse) r
esponse;
68:             if (!(httpRequest == null || httpResponse == null)) {
69:                 if (httpRequest.getPathInfo() != null && httpReque
st.getPathInfo().matches("^/cart(/)?"))
70:                     filterCart(httpRequest, httpResponse, chai
n);
71:             else
72:                 filterItem(httpRequest, httpResponse, chai
n);
73:             } else {
74:                 chain.doFilter(request, response);
75:             }
76:         } catch (Exception e) {
77:             System.out.println(e.getMessage());
78:             e.printStackTrace();
79:             chain.doFilter(request, response);
80:         }
81:     }
82:
83:     /**
84:      * @see Filter#init(FilterConfig)
85:      */
86:     public void init(FilterConfig fConfig) throws ServletException {
87:         // TODO Auto-generated method stub
88:     }
89:
90:     private void filterCart(HttpServletRequest request, HttpServletResponse re
sponse, FilterChain chain) throws IOException, ServletException {
91:         if (request.getMethod().equals("POST")) {
92:             try {
93:                 int quantity = Integer.parseInt(request.getParamet
er("qty"));
94:                 String number = (String) request.getParameter("ite
m");
95:                 if (!number.equals("")) {
96:                     Map<String, String> activeCrossSells = get
CrossSellMap(request);
97:                     if (quantity > 0) { // Added to cart
98:                         System.out.println("Added to cart
moment");
99:                         if (TO_CROSS_SELL.get(number) != n
ull)
100:                             activeCrossSells.put(numbe
r, TO_CROSS_SELL.get(number));
101:                     } else {
102:                         System.out.println("Removed from c
art moment");
103:                         activeCrossSells.remove(number);
104:                     }
105:                     setCrossSellMap(request, activeCrossSells)
;
106:                 } else {
107:                     throw new Exception("Empty item number sen
t from client");
108:                 }
109:             } catch (Exception e) {
110:                 System.out.println(e.getMessage());
111:                 e.printStackTrace();
112:             }
113:         }
114:         chain.doFilter(request, response);
115:     }
116:
117:     private void filterItem(HttpServletRequest request, HttpServletResponse re
sponse, FilterChain chain) throws IOException, ServletException {
118:         if (request.getMethod().equals("GET")) {
119:             try {
120:                 CartItem ci = (CartItem) request.getAttribute("car

```

```

tItem");
121:                                     EFoods model = (EFoods) request.getServletContext(
).getAttribute(ctrl.Front.MODEL_KEY);
122:
123:                                     Map<String, String> activeCrossSells = getCrossSel
lMap(request);
124:
125:                                     if (ci != null && activeCrossSells != null) {
126:                                         String crossSellThis = activeCrossSells.ge
t(ci.getItem().getNumber());
127:                                         if (crossSellThis != null) {
128:                                             ItemBean crossSellItem = model.ite
ms(crossSellThis, ItemDAO.CAT_ALL, ItemDAO.PAGE_ALL, ItemDAO.LIMIT_ALL, ItemDAO.NO_FILTER
).get(0);
129:                                             request.setAttribute("crossSellIte
m", crossSellItem);
130:                                             RequestDispatcher rd = request.get
RequestDispatcher("/partials/_crossSellItem.jspx");
131:                                             if (request.getDispatcherType().eq
uals(DispatcherType.FORWARD))
132:                                                 rd.forward(request, respon
se);
133:                                             else if (request.getDispatcherType
().equals(DispatcherType.INCLUDE))
134:                                                 rd.include(request, respon
se);
135:                                             return;
136:                                         }
137:                                     }
138:                                     } catch (Exception e) {
139:                                         System.out.println(e.getMessage());
140:                                         e.printStackTrace();
141:                                     }
142:                                 }
143:                                 chain.doFilter(request, response);
144:                             }
145:
146:                             private Map<String,String> getCrossSellMap(HttpServletRequest request) {
147:                                 Map<String, String> csm = (HashMap<String,String>) request
148:                                     .getSession().getAttribute(ACTIVE_CROSS_SELLS_MAP_
KEY);
149:                                 return csm == null ? new HashMap<String, String>() : csm;
150:                             }
151:
152:                             private void setCrossSellMap(HttpServletRequest request, Map<String,String
> csm) {
153:                                 request.getSession().setAttribute(ACTIVE_CROSS_SELLS_MAP_KEY, csm)
;
154:                             }
155:
156: }

```

```

1: package listener;
2:
3: import javax.servlet.annotation.WebListener;
4: import javax.servlet.http.HttpSessionAttributeListener;
5: import javax.servlet.http.HttpSessionBindingEvent;
6:
7: /**
8:  * Application Lifecycle Listener implementation class TimeStatistic
9:  *
10:  */
11: @WebListener
12: public class TimeStatistician implements HttpSessionAttributeListener {
13:
14:     /**
15:      * Default constructor.
16:      */
17:     public TimeStatistician() {
18:         // TODO Auto-generated constructor stub
19:     }
20:
21:     /**
22:      * @see HttpSessionAttributeListener#attributeRemoved(HttpSessionBindingEvent)
23:      */
24:     public void attributeRemoved(HttpSessionBindingEvent se) {
25:         // DO NOTHING!
26:     }
27:
28:     /**
29:      * @see HttpSessionAttributeListener#attributeAdded(HttpSessionBindingEvent)
30:      */
31:     public void attributeAdded(HttpSessionBindingEvent se) {
32:         if (se.getName().equals("cartItemsCount"))
33:         {
34:             long frsVstTime = se.getSession().getCreationTime(); //
fresh visit time
35:             long curTime = System.currentTimeMillis(); // current time (w
hen first item added)
36:
37:             updateAvgAddItemTime(se, curTime - frsVstTime);
38:
39:             se.getSession().setAttribute("cartLastManipTime", curTime);
40:         }
41:         else if (se.getName().equals("checkedOut")) // when client has checked
out
42:         {
43:             long frsVstTime = se.getSession().getCreationTime(); //
fresh visit time
44:             long curTime = System.currentTimeMillis(); // current time (w
hen client checked out)
45:
46:             updateAvgCheckOutTime(se, curTime - frsVstTime);
47:         }
48:     }
49:
50:     /**
51:      * @see HttpSessionAttributeListener#attributeReplaced(HttpSessionBindingEvent)
52:      */
53:     public void attributeReplaced(HttpSessionBindingEvent se) {
54:         if (se.getName().equals("cartItemsCount"))
55:         {
56:             long lstMnpTime = (long) se.getSession().getAttribute("cartLastMa
nipTime"); // item last added time
57:             long curTime = System.currentTimeMillis(); // current time
58:             int oldCount = (int) se.getValue();
59:             int newCount = (int) se.getSession().getAttribute("cartItemsCount
");
60:
61:             // when there is item added to the cart
62:             if (newCount > oldCount)

```

```
63:         {
64:             updateAvgAddItemTime(se, curTime - lstMnpTime);
65:             se.getSession().setAttribute("cartLastManipTime", curTime
);
66:         }
67:     }
68: }
69:
70: // Do not remove static, since we need class-wide synchronization
71: private static synchronized void updateAvgAddItemTime(HttpSessionBindingEvent
evn, long duration)
72: {
73:     if (evn.getSession().getServletContext().getAttribute("avgAddItemTime") =
= null)
74:     {
75:         // when the very first item added
76:         evn.getSession().getServletContext().setAttribute("avgAddItemTime
", duration);
77:         evn.getSession().getServletContext().setAttribute("NItemsAdded",
(long) 1);
78:     }
79:     else
80:     {
81:         long avgTime = (long) evn.getSession().getServletContext().getAtt
ribute("avgAddItemTime");
82:         long N = (long) evn.getSession().getServletContext().getAttribute
("NItemsAdded");
83:
84:         // cast to double for computation
85:         double durAcc = (new Long(duration)).doubleValue();
86:         double avgTimeAcc = (new Long(avgTime)).doubleValue();
87:         double NAcc = (new Long(N)).doubleValue();
88:         // update statistic
89:         double newAvgTime = (NAcc / (NAcc + 1.0)) * avgTimeAcc + (1.0 / (
NAcc + 1.0)) * durAcc;
90:
91:         evn.getSession().getServletContext().setAttribute("avgAddItemTime
", (new Double(newAvgTime)).longValue());
92:         evn.getSession().getServletContext().setAttribute("NItemsAdded",
N + 1);
93:     }
94: }
95:
96: // Do not remove static, since we need class-wide synchronization
97: private static synchronized void updateAvgCheckOutTime(HttpSessionBindingEvent
evn, long duration)
98: {
99:     if (evn.getSession().getServletContext().getAttribute("avgCheckOutTime")
== null)
100:     {
101:         // when the very order placed
102:         evn.getSession().getServletContext().setAttribute("avgCheckOutTim
e", duration);
103:         evn.getSession().getServletContext().setAttribute("NCheckedOut",
(long) 1);
104:     }
105:     else
106:     {
107:         long avgTime = (long) evn.getSession().getServletContext().getAtt
ribute("avgCheckOutTime");
108:         long N = (long) evn.getSession().getServletContext().getAttribute
("NCheckedOut");
109:
110:         // cast to double for computation
111:         double durAcc = (new Long(duration)).doubleValue();
112:         double avgTimeAcc = (new Long(avgTime)).doubleValue();
113:         double NAcc = (new Long(N)).doubleValue();
114:         // update statistic
115:         double newAvgTime = (NAcc / (NAcc + 1.0)) * avgTimeAcc + (1.0 / (
NAcc + 1.0)) * durAcc;
```

```

116:
117:         evn.getSession().getServletContext().setAttribute("avgCheckOutTime", (new Double(newAvgTime)).longValue());
118:         evn.getSession().getServletContext().setAttribute("NCheckedOut",
N + 1);
119:     }
120: }
121: }
```

```

1: package model;
2:
3: import java.sql.Connection;
4: import java.sql.PreparedStatement;
5: import java.util.Iterator;
6: import java.util.List;
7: import java.util.Queue;
8:
9: public class BaseDAO {
10:
11:
12:     public final static int ID_ALL = -1;
13:     public final static int PAGE_ALL = 1;
14:     public final static int LIMIT_ALL = -1;
15:     public BaseDAO() {
16:         // TODO Auto-generated constructor stub
17:     }
18:
19:     protected class PrepareInstruction {
20:         static final int TYPE_STRING = 0x0a;
21:         static final int TYPE_INT = 0x0b;
22:         int type;
23:         Object value;
24:         public PrepareInstruction(int type, Object value) {
25:             super();
26:             this.type = type;
27:             this.value = value;
28:         }
29:     }
30:
31:     /**
32:      * Creates a new string based a list of where clauses, this list
33:      * is joined by the operator.
34:      *
35:      * @param wheres
36:      * @param operator
37:      * @return A string joined by the passed operator
38:      */
39:     protected String createWhereString(List<String> wheres, String operator) {
40:         String whereString = "";
41:         Iterator<String> iter = wheres.iterator();
42:         whereString += iter.hasNext() ? iter.next() : "";
43:         while (iter.hasNext()) whereString += " " + operator + " " + iter.next();
44:         return "(" + whereString + ")";
45:     }
46:
47:     /**
48:      *
49:      * @param query
50:      * @param piQ
51:      * @param connection
52:      * @return The prepared statement.
53:      * @throws Exception
54:      */
55:     protected PreparedStatement instaPrepareStatement(String query, Queue<Prep
areInstruction> piQ, Connection connection) throws Exception {
56:         PreparedStatement ps = connection.prepareStatement(query);
57:         int idx=0;
58:         while (piQ.size() > 0) {
59:             PrepareInstruction pi = piQ.poll();
60:             idx++; //Start with 1
61:             switch (pi.type) {
62:                 case PrepareInstruction.TYPE_STRING:
63:                     ps.setString(idx, (String) pi.value);
64:                     break;
65:                 case PrepareInstruction.TYPE_INT:
66:                     ps.setInt(idx, (int) pi.value);
67:                     break;
68:                 default:
69:                     throw new Exception("Instructions are invalid");

```

```
70:         }
71:     }
72:     return ps;
73: }
74:
75: /**
76:  * Creates a string for paging. This is used to fetch
77:  * a specific amount of rows from the database.
78:  *
79:  * @param page
80:  * @param limit
81:  * @return
82:  */
83: protected String pagingString(int page, int limit) {
84:     String ret = "";
85:
86:     if (page > 0) {
87:         ret += " OFFSET " + (page - 1) * limit + " ROWS ";
88:         if (limit != LIMIT_ALL) {
89:             ret += " FETCH NEXT " + limit + " ROWS ONLY";
90:         }
91:     }
92:
93:     return ret;
94: }
95: }
```

```
1: package model;
2:
3: import javax.xml.bind.annotation.XmlAttribute;
4: import javax.xml.bind.annotation.XmlElement;
5:
6: import model.CartModel.CartItem;
7:
8: public class CartItemBean {
9:
10:     @XmlAttribute
11:     String number;
12:
13:     @XmlElement
14:     String name;
15:
16:     @XmlElement
17:     float price;
18:
19:     @XmlElement
20:     int quantity;
21:
22:     @XmlElement
23:     String extended;
24:
25:     public CartItemBean() {
26:         // TODO Auto-generated constructor stub
27:     }
28:     public CartItemBean(CartItem cartItem) {
29:         this.number = cartItem.getItem().getNumber();
30:         this.name = cartItem.getItem().getName();
31:         this.price = cartItem.getItem().getPrice();
32:         this.quantity = cartItem.getQuantity();
33:         this.extended = cartItem.getItemTotal();
34:     }
35:
36: }
```



```
1: package model;
2:
3: import java.util.ArrayList;
4: import java.util.List;
5:
6: import javax.xml.bind.annotation.XmlElement;
7: import javax.xml.bind.annotation.XmlRootElement;
8:
9: import model.CartModel.CartItem;
10:
11: @XmlRootElement
12: public class CartItemsWrapper
13: {
14:     @XmlElement(name="item")
15:     private List<CartItemBean> itemsList;
16:
17:     public CartItemsWrapper()
18:     {
19:         super();
20:         // TODO Auto-generated constructor stub
21:     }
22:
23:     public CartItemsWrapper(List<CartItem> cartItems)
24:     {
25:         List<CartItemBean> cartItemsList = new ArrayList<CartItemBean>();
26:         for(CartItem ci: cartItems)
27:             cartItemsList.add(new CartItemBean(ci));
28:
29:         this.itemsList = cartItemsList;
30:     }
31: }
```

```

1: package model;
2:
3: import java.math.BigDecimal;
4: import java.math.MathContext;
5: import java.math.RoundingMode;
6: import java.util.ArrayList;
7: import java.util.Iterator;
8: import java.util.LinkedHashMap;
9: import java.util.List;
10: import java.util.Map;
11:
12: import javax.servlet.http.HttpServletRequest;
13:
14: import util.SSOAuthenticator;
15:
16: public class CartModel {
17:
18:     public class CartItem {
19:         ItemBean item;
20:         int quantity;
21:
22:         /**
23:          * @return the item
24:          */
25:         public ItemBean getItem() {
26:             return item;
27:         }
28:
29:         /**
30:          * @return the quantity
31:          */
32:         public int getQuantity() {
33:             return quantity;
34:         }
35:
36:         public CartItem(ItemBean item, int quantity) {
37:             super();
38:             this.item = item;
39:             this.quantity = quantity;
40:         }
41:
42:         public String getItemTotal() {
43:             return (new BigDecimal(Float.toString(item.price))).multip
44: ly(new BigDecimal(quantity)).toString();
45:         }
46:     }
47:
48:     private static final BigDecimal TAX_MULTIPLIER = new BigDecimal("0.13");
49:     private static final BigDecimal FREE_SHIPPING_LIMIT = new BigDecimal("100.
50: 00");
51:     private static final BigDecimal SHIPPING_CHARGE = new BigDecimal("5.00");
52:     private static final String CART_SESSION_KEY = "com.eFoods.model.Cart";
53:
54:     private List<CartItem> cartItems;
55:     private UserBean account;
56:     private BigDecimal total;
57:     private BigDecimal shipping;
58:     private BigDecimal tax;
59:
60:     public CartModel(UserBean account) {
61:         super();
62:         this.cartItems = new ArrayList<CartItem>();
63:         this.account = account;
64:         this.total = BigDecimal.ZERO;
65:         this.shipping = SHIPPING_CHARGE;
66:         this.tax = BigDecimal.ZERO;
67:     }
68:
69:     public synchronized void manipulateCart(ItemBean item, int qty)

```

```

69:         throws Exception {
70:     try {
71:         for (Iterator<CartItem> i = cartItems.listIterator(); i.hasNext(); ) {
72:             CartItem ci = i.next();
73:             if (ci.item.equals(item)) {
74:                 if (qty < 1) {
75:                     i.remove();
76:                     System.out.println("Cart\t: Item removed. "
77:                                     + item.number);
78:                 } else {
79:                     ci.quantity = qty;
80:                     System.out.println("Cart\t: Item quantity updated to " + qty
81:                                     + " for " + item.number);
82:                 }
83:                 return;
84:             }
85:         }
86:         if (qty > 0) {
87:             cartItems.add(new CartItem(item, qty));
88:             System.out.println("Cart\t: Item added. " + qty +
89:                             " of "
90:                             + item.number);
91:         } else {
92:             throw new Exception(
93:                 "Invalid operation. Cannot set qty to 0 when it doesn't exist, NOOB");
94:         }
95:     } finally {
96:         computeCart();
97:     }
98: }
99:
100:
101: private void computeCart() {
102:     this.total = calculateTotal();
103:     this.shipping = calculateShipping();
104:     this.tax = calculateTax();
105:
106:     System.out.println("Cart : Refreshed. Total:" + total + " Shipping
107:                       + shipping + " Tax:" + tax);
108: }
109:
110: private BigDecimal calculateShipping() {
111:     if (isFreeShipping())
112:         return round(BigDecimal.ZERO, 4);
113:     return round(SHIPPING_CHARGE, 4);
114: }
115:
116: private BigDecimal calculateTotal() {
117:     BigDecimal t = BigDecimal.ZERO;
118:     for (CartItem ci : cartItems) {
119:         System.out.println(new BigDecimal(ci.item.getPrice()));
120:         System.out.println(new BigDecimal(ci.quantity));
121:         t = t.add(new BigDecimal(ci.item.getPrice()).multiply(
122:             new BigDecimal(ci.quantity)));
123:         System.out.println(t);
124:     }
125:     System.out.println(round(t,4));
126:     return round(t, 4);
127: }
128:
129: private BigDecimal calculateTax() {
130:     return round((isFreeShipping() ? total.multiply(TAX_MULTIPLIER) :
total.add(shipping).multiply(TAX_MULTIPLIER)), 3);

```

```
131:     }
132:
133:     /**
134:     *
135:     * @param number
136:     * @param places
137:     * @return rounded BigDecimal to passed number of places.
138:     */
139:     private BigDecimal round(BigDecimal number, int places) {
140:         return number.round(new MathContext(places, RoundingMode.HALF_UP))
;
141:     }
142:
143:     public boolean isFreeShipping() {
144:         return (total.compareTo(FREE_SHIPPING_LIMIT) >= 0);
145:     }
146:
147:     /**
148:     * @param cartItems
149:     *         the cartItems to set
150:     */
151:     public List<CartItem> getCartItems() {
152:         return cartItems;
153:     }
154:
155:     /**
156:     * Gets the CartItem object for the given item
157:     *
158:     * @param item
159:     * @return corresponding CartItem if found, null if doesnt exist in cart.
160:     */
161:     public CartItem getCartItemFor(ItemBean item) {
162:         for (CartItem ci : cartItems)
163:             if (ci.item.equals(item))
164:                 return ci;
165:         return null;
166:     }
167:
168:     /**
169:     * Creates a map of cart itesm so that it could be accessed directly from
item number
170:     * @return map containing item number as key and cartitem as value
171:     */
172:     public Map<String, CartItem> mappedCartItems() {
173:         Map<String, CartItem> map = new LinkedHashMap<String, CartItem>();
174:         for (CartItem i : this.getCartItems())
175:             map.put(i.getItem().getName(), i);
176:         return map;
177:     }
178:
179:     /**
180:     * @param wot
181:     */
182:     public void setCartItems(List<CartItem> cartItems) {
183:         this.cartItems = cartItems;
184:     }
185:
186:     /**
187:     * @return the account
188:     */
189:     public UserBean getAccount() {
190:         return account;
191:     }
192:
193:     /**
194:     * @param account
195:     *         the account to set
196:     */
197:     public void setAccount(UserBean account) {
198:         if ((this.account == null || !this.account.equals(account)) && acc
```

```

ount != null)
199:         System.out.println("CartModel: Ownership changed to "
200:                               + account.name);
201:         this.account = account;
202:     }
203:
204:     /**
205:      * @return the total
206:      */
207:     public BigDecimal getTotal() {
208:         return total;
209:     }
210:
211:     /**
212:      * @param total
213:      *      the total to set
214:      */
215:     public void setTotal(BigDecimal total) {
216:         this.total = total;
217:     }
218:
219:     /**
220:      * @return the shipping
221:      */
222:     public BigDecimal getShipping() {
223:         return shipping;
224:     }
225:
226:     /**
227:      * @param shipping
228:      *      the shipping to set
229:      */
230:     public void setShipping(BigDecimal shipping) {
231:         this.shipping = shipping;
232:     }
233:
234:     /**
235:      * @return the tax
236:      */
237:     public BigDecimal getTax() {
238:         return tax;
239:     }
240:
241:     /**
242:      * @param tax
243:      *      the tax to set
244:      */
245:     public void setTax(BigDecimal tax) {
246:         this.tax = tax;
247:     }
248:
249:     public static CartModel getInstance(HttpServletRequest request,
250:                                       SSOAuthenticator auth) {
251:         CartModel dis;
252:         if ((dis = (CartModel) request.getSession().getAttribute(
253:             CART_SESSION_KEY)) == null) {
254:             synchronized (CartModel.class) {
255:                 if ((dis = (CartModel) request.getSession().getAtt
256:                     ribute(
257:                         CART_SESSION_KEY)) == null) {
258:                     dis = new CartModel(null);
259:                     request.getSession().setAttribute(CART_SES
260: SION_KEY, dis);
261:                     System.out.println("Cart: Initialized");
262:                 }
263:             }
264:             dis.setAccount(auth.getUser(request));
265:             return dis;

```

```
266:         }
267:
268:     public void clear() {
269:         this.cartItems = new ArrayList<CartItem>();
270:         computeCart();
271:     }
272:
273:     /**
274:      *
275:      * @return The grandTotal is cumulative amount of tax + shipping + total
276:      */
277:     public BigDecimal getGrandTotal() {
278:         return tax.add(total).add(shipping);
279:     }
280:
281: }
```

```

1: package model;
2:
3: public class CategoryBean {
4:     private int id;
5:     private String name;
6:     private String description;
7:     private byte[] picture;
8:
9:
10:    public CategoryBean(int id, String name, String description, byte[] picture) {
11:        super();
12:        this.id = id;
13:        this.name = name;
14:        this.description = description;
15:        this.picture = picture;
16:    }
17:
18:
19:    /**
20:     * @return the id
21:     */
22:    public int getId() {
23:        return id;
24:    }
25:
26:    /**
27:     * @param id the id to set
28:     */
29:    public void setId(int id) {
30:        this.id = id;
31:    }
32:
33:    /**
34:     * @return the name
35:     */
36:    public String getName() {
37:        return name;
38:    }
39:
40:    /**
41:     * @param name the name to set
42:     */
43:    public void setName(String name) {
44:        this.name = name;
45:    }
46:
47:    /**
48:     * @return the description
49:     */
50:    public String getDescription() {
51:        return description;
52:    }
53:
54:    /**
55:     * @param description the description to set
56:     */
57:    public void setDescription(String description) {
58:        this.description = description;
59:    }
60:
61:    /**
62:     * @return the picture
63:     */
64:    public byte[] getPicture() {
65:        return picture;
66:    }
67:
68:    /**
69:     * @param picture the picture to set

```

```
70:         */
71:         public void setPicture(byte[] picture) {
72:             this.picture = picture;
73:         }
74:
75:
76: }
```



```

1: package model;
2:
3: import java.math.BigInteger;
4: import java.sql.Connection;
5: import java.sql.PreparedStatement;
6: import java.sql.ResultSet;
7: import java.util.ArrayList;
8: import java.util.List;
9:
10:
11: import util.eFoodsDataSource;
12: public class CategoryDAO extends BaseDAO {
13:
14:
15:
16:     public CategoryDAO() {}
17:
18:     public List<CategoryBean> retrieve() throws Exception {
19:         return retrieve(ID_ALL);
20:     }
21:
22:
23:
24:
25:     public List<CategoryBean> retrieve(int id) throws Exception {
26:         Connection connection = null;
27:         PreparedStatement preparedStatement;
28:         ResultSet resultSet;
29:         List<CategoryBean> retval = new ArrayList<CategoryBean>();
30:
31:
32:
33:         try {
34:             connection = eFoodsDataSource.getConnection();
35:
36:             if(id==ID_ALL) {
37:                 String query = "SELECT id, name, description, pict
ure FROM category";
38:                 preparedStatement = connection.prepareStatement(qu
ery);
39:             } else {
40:                 String query = "SELECT id, name, description, pict
ure FROM category WHERE id = ?";
41:                 preparedStatement = connection.prepareStatement(qu
ery);
42:                 preparedStatement.setInt(1, id);
43:             }
44:
45:             resultSet = preparedStatement.executeQuery();
46:
47:             while (resultSet.next()) {
48:                 retval.add(new CategoryBean(resultSet.getInt("id")
,
49:                 resultSet.getString("name"),
50:                 resultSet.getString("description")
,
51:                 new BigInteger(resultSet.getString
("picture"), 16).toByteArray()));
52:             }
53:         } catch (Exception e) {
54:             throw e;
55:         } finally {
56:             if (connection != null)
57:                 connection.close();
58:         }
59:
60:         return retval;
61:     }
62:
63: }

```

```

1: package model;
2:
3: import java.io.FileWriter;
4: import java.io.StringWriter;
5: import java.util.List;
6:
7: import javax.xml.bind.JAXBContext;
8: import javax.xml.bind.Marshaller;
9: import javax.xml.transform.stream.StreamResult;
10:
11: import util.PurchaseOrderFiles;
12:
13: public class EFoods {
14:     private CategoryDAO categoryDAO;
15:     private ItemDAO itemDAO;
16:     public EFoods() {
17:         categoryDAO = new CategoryDAO();
18:         itemDAO = new ItemDAO();
19:     }
20:
21:     public List<CategoryBean> categories(int id) throws Exception {
22:         return categoryDAO.retrieve(id);
23:     }
24:
25:     public List<ItemBean> items(String number, int catId, int page, int limit,
String filter) throws Exception {
26:         return itemDAO.retrieve(number, catId, page, limit, filter);
27:     }
28:
29:     /**
30:      * Creates a purchase order based on the current items in the cart.
31:      *
32:      * @param cart
33:      * @param filepath
34:      * @return the purchase order id
35:      * @throws Exception
36:      */
37:     public synchronized int createPurchaseOrder(CartModel cart, String filepath
h) throws Exception {
38:         PurchaseOrderFiles pofs = new PurchaseOrderFiles(filepath);
39:         int orderId = pofs.getNextOrderId();
40:         PurchaseOrderWrapper poWrapper = new PurchaseOrderWrapper(cart, or
derId);
41:         StringWriter sWriter = null;
42:         FileWriter fWriter = null;
43:         try {
44:             JAXBContext jaxbCtx = JAXBContext.newInstance(poWrapper
.getClass());
45:             Marshaller marshaller = jaxbCtx.createMarshaller();
46:
47:             //marshaller.setProperty(Marshaller.JAXB_SCHEMA_LOCATION,
48:             // "SIS.xsd");
49:             marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
50:             Boolean.TRUE);
51:             marshaller.setProperty(Marshaller.JAXB_FRAGMENT, Boolean.T
RUE);
52:
53:             sWriter = new StringWriter();
54:             sWriter.write("<?xml version='1.0'?>");
55:             sWriter.write("\n");
56:             sWriter.write("<?xml-stylesheet type='text/xsl' href='Purc
haseOrders.xsl' ?>");
57:
58:             sWriter.write("\n");
59:             marshaller.marshal(poWrapper, new StreamResult(sWriter));
60:
61:             pofs.storeNewOrder(orderId, cart.getAccount().getName(), s
Writer.toString());
62:             return orderId;
63:         } catch (Exception e) {
64:             throw e;

```

```
65:                } finally {
66:                    if (fWriter != null)
67:                        fWriter.close();
68:                    if (sWriter != null)
69:                        sWriter.close();
70:                }
71:
72:            }
73:
74:
75:
76:
77:
78: }
```

```
1: package model;
2:
3: public class ItemBean {
4:
5:     String number;
6:     String name;
7:     float price;
8:     int qty;
9:     int onOrder;
10:    int reOrder;
11:    CategoryBean category;
12:    int supplierId;
13:    int costPrice;
14:    String unit;
15:
16:    public ItemBean(String number, String name, float price, int qty, int onOr
der,
17:                    int reOrder, CategoryBean category, int supplierId, int co
stPrice,
18:                    String unit) {
19:        super();
20:        this.number = number;
21:        this.name = name;
22:        this.price = price;
23:        this.qty = qty;
24:        this.onOrder = onOrder;
25:        this.reOrder = reOrder;
26:        this.category = category;
27:        this.supplierId = supplierId;
28:        this.costPrice = costPrice;
29:        this.unit = unit;
30:    }
31:
32:    /**
33:     * @return the number
34:     */
35:    public String getNumber() {
36:        return number;
37:    }
38:
39:    /**
40:     * @param number the number to set
41:     */
42:    public void setNumber(String number) {
43:        this.number = number;
44:    }
45:
46:    /**
47:     * @return the name
48:     */
49:    public String getName() {
50:        return name;
51:    }
52:
53:    /**
54:     * @param name the name to set
55:     */
56:    public void setName(String name) {
57:        this.name = name;
58:    }
59:
60:    /**
61:     * @return the price
62:     */
63:    public float getPrice() {
64:        return price;
65:    }
66:
67:    /**
68:     * @param price the price to set
```

```
69:         */
70:         public void setPrice(float price) {
71:             this.price = price;
72:         }
73:
74:         /**
75:          * @return the qty
76:          */
77:         public int getQty() {
78:             return qty;
79:         }
80:
81:         /**
82:          * @param qty the qty to set
83:          */
84:         public void setQty(int qty) {
85:             this.qty = qty;
86:         }
87:
88:         /**
89:          * @return the onOrder
90:          */
91:         public int getOnOrder() {
92:             return onOrder;
93:         }
94:
95:         /**
96:          * @param onOrder the onOrder to set
97:          */
98:         public void setOnOrder(int onOrder) {
99:             this.onOrder = onOrder;
100:        }
101:
102:        /**
103:         * @return the reOrder
104:         */
105:        public int getReOrder() {
106:            return reOrder;
107:        }
108:
109:        /**
110:         * @param reOrder the reOrder to set
111:         */
112:        public void setReOrder(int reOrder) {
113:            this.reOrder = reOrder;
114:        }
115:
116:        /**
117:         * @return the category
118:         */
119:        public CategoryBean getCategory() {
120:            return category;
121:        }
122:
123:        /**
124:         * @param category the category to set
125:         */
126:        public void setCategory(CategoryBean category) {
127:            this.category = category;
128:        }
129:
130:        /**
131:         * @return the supplierId
132:         */
133:        public int getSupplierId() {
134:            return supplierId;
135:        }
136:
137:        /**
138:         * @param supplierId the supplierId to set
```

```
139:         */
140:     public void setSupplierId(int supplierId) {
141:         this.supplierId = supplierId;
142:     }
143:
144:     /**
145:      * @return the costPrice
146:      */
147:     public int getCostPrice() {
148:         return costPrice;
149:     }
150:
151:     /**
152:      * @param costPrice the costPrice to set
153:      */
154:     public void setCostPrice(int costPrice) {
155:         this.costPrice = costPrice;
156:     }
157:
158:     /**
159:      * @return the unit
160:      */
161:     public String getUnit() {
162:         return unit;
163:     }
164:
165:     /**
166:      * @param unit the unit to set
167:      */
168:     public void setUnit(String unit) {
169:         this.unit = unit;
170:     }
171:
172:     /* (non-Javadoc)
173:      * @see java.lang.Object#hashCode()
174:      */
175:     @Override
176:     public int hashCode() {
177:         final int prime = 31;
178:         int result = 1;
179:         result = prime * result + ((number == null) ? 0 : number.hashCode(
180:     ));
181:         return result;
182:     }
183:
184:     /* (non-Javadoc)
185:      * @see java.lang.Object#equals(java.lang.Object)
186:      */
187:     @Override
188:     public boolean equals(Object obj) {
189:         if (this == obj) {
190:             return true;
191:         }
192:         if (obj == null) {
193:             return false;
194:         }
195:         if (!(obj instanceof ItemBean)) {
196:             return false;
197:         }
198:         ItemBean other = (ItemBean) obj;
199:         if (number == null) {
200:             if (other.number != null) {
201:                 return false;
202:             }
203:         } else if (!number.equals(other.number)) {
204:             return false;
205:         }
206:         return true;
207:     }
```

208: }

```

1: package model;
2:
3: import java.sql.Connection;
4: import java.sql.PreparedStatement;
5: import java.sql.ResultSet;
6: import java.util.ArrayList;
7: import java.util.LinkedList;
8: import java.util.List;
9: import java.util.Queue;
10:
11: import util.eFoodsDataSource;
12:
13: public class ItemDAO extends BaseDAO {
14:
15:     public final static String NUMBER_ALL = "-1";
16:     public final static int CAT_ALL = -1;
17:     public final static String NO_FILTER = "__!!__";
18:
19:     public static final String BASE_QUERY = "SELECT I.number, "
20:         + "I.name, I.price, I.qty, I.onorder, "
21:         + "I.reorder, I.catid, I.supid, I.costprice, "
22:         + "I.unit, C.id as \"CAT_ID\", C.name as \"CAT_NAME\", "
23:         + "C.description as \"CAT_DESCRIPTION\" FROM ITEM I "
24:         + "INNER JOIN CATEGORY C ON I.catId = C.id";
25:
26:     public ItemDAO() {}
27:
28:     /**
29:      * Retrieves all the items from the database.
30:      * @return A List containing all the items.
31:      * @throws Exception
32:      */
33:     public List<ItemBean> retrieve() throws Exception {
34:         return retrieve(NUMBER_ALL);
35:     }
36:
37:
38:     /**
39:      * Retrieves a single item from the database.
40:      * @param number
41:      * @return A List containing the single item.
42:      * @throws Exception
43:      */
44:     public List<ItemBean> retrieve(String number) throws Exception {
45:         return retrieve(number, CAT_ALL, PAGE_ALL, LIMIT_ALL, NO_FILTER);
46:     }
47:
48:     /**
49:      * Creates a query for the DB based on the passed parameters.
50:      *
51:      * @param number
52:      * @param catId
53:      * @param page
54:      * @param limit
55:      * @param filter
56:      * @return The list of the items passed on the generated query.
57:      * @throws Exception
58:      */
59:     public List<ItemBean> retrieve(String number, int catId, int page, int lim
it, String filter) throws Exception {
60:         Connection connection = null;
61:         PreparedStatement preparedStatement;
62:         ResultSet rs;
63:         List<ItemBean> retval = new ArrayList<ItemBean>();
64:
65:         try {
66:             connection = eFoodsDataSource.getConnection();
67:
68:
69:             Queue<PrepareInstruction> instructions=new LinkedL

```



```

ist<PrepareInstruction>();
70:                                     List<String> wheres = new ArrayList<String>();
71:
72:                                     if (!(number.equals(NUMBER_ALL))) {
73:                                         wheres.add("I.number = ?");
74:                                         instructions.add(new PrepareInstruction(Pr
epareInstruction.TYPE_STRING, number));
75:                                     }
76:                                     if (catId != ID_ALL) {
77:                                         wheres.add("C.id = ?");
78:                                         instructions.add(new PrepareInstruction(Pr
epareInstruction.TYPE_INT, catId));
79:                                     }
80:
81:                                     if (!filter.equals(NO_FILTER)) {
82:                                         wheres.add("LOWER(I.name) LIKE LOWER( ? )"
);
83:                                         instructions.add(new PrepareInstruction(Pr
epareInstruction.TYPE_STRING, "%" + filter + "%"));
84:                                     }
85:                                     String query = BASE_QUERY;
86:                                     if (wheres.size() > 0) { query += " WHERE " + crea
teWhereString(wheres, "AND"); }
87:
88:                                     query += pagingString(page, limit);
89:                                     System.out.println("DB Query: " + query.replaceFirst(BASE_
QUERY, ""));
90:                                     preparedStatement = instaPrepareStatement(query, i
nstructions, connection);
91:
92:
93:
94:                                     rs = preparedStatement.executeQuery();
95:
96:                                     while (rs.next()) {
97:                                         CategoryBean cat = new CategoryBean(rs.getInt("cat
_id"),
98:                                             rs.getString("cat_name"),
99:                                             rs.getString("cat_description"), n
ull);
100:                                         ItemBean item = new ItemBean(rs.getString("number"
),
101:                                             rs.getString("name"), rs.getFloat(
"price"),
102:                                             rs.getInt("qty"), rs.getInt("onord
er"),
103:                                             rs.getInt("reorder"), cat, rs.getI
nt("supid"),
104:                                             rs.getInt("costprice"), rs.getStri
ng("unit"));
105:                                         retval.add(item);
106:                                     }
107:                                     } catch (Exception e) {
108:                                         throw e;
109:                                     } finally {
110:                                         if (connection != null)
111:                                             connection.close();
112:                                     }
113:
114:                                     return retval;
115:                                 }
116:
117: }

```

```
1: package model;
2:
3: import java.util.List;
4:
5: import javax.xml.bind.annotation.XmlElement;
6: import javax.xml.bind.annotation.XmlRootElement;
7:
8: @XmlRootElement(name="PurchaseOrderFiles")
9: public class POFilesWrapper {
10:
11:     @XmlElement(name="PurchaseOrderFile")
12:     List<POFileWrapper> pofList;
13:     public POFilesWrapper() {
14:         // TODO Auto-generated constructor stub
15:     }
16:
17:     public POFilesWrapper(List<POFileWrapper> pofList) {
18:         this.pofList = pofList;
19:     }
20:
21: }
```

```
1: package model;
2:
3: import javax.xml.bind.annotation.XmlElement;
4:
5: public class POFileWrapper {
6:     @XmlElement
7:     String url;
8:
9:     @XmlElement
10:    int id;
11:
12:    /**
13:     * @return the url
14:     */
15:    public String getUrl() {
16:        return url;
17:    }
18:    /**
19:     * @return the id
20:     */
21:    public int getId() {
22:        return id;
23:    }
24:
25:    public POFileWrapper() {
26:        // TODO Auto-generated constructor stub
27:    }
28:
29:    public POFileWrapper(String url, int i) {
30:        super();
31:        this.url = url;
32:        this.id = i;
33:
34:    }
35: }
```

```
1: package model;
2: import java.util.Date;
3:
4: import javax.xml.bind.annotation.XmlAttribute;
5: import javax.xml.bind.annotation.XmlElement;
6: import javax.xml.bind.annotation.XmlRootElement;
7: import javax.xml.bind.annotation.XmlSchemaType;
8:
9: import model.CartModel;
10:
11: @XmlRootElement(name="order")
12: public class PurchaseOrderWrapper {
13:
14:     @XmlAttribute
15:     private int id;
16:
17:     @XmlAttribute
18:     @XmlSchemaType(name="date")
19:     Date submitted;
20:
21:     @XmlElement
22:     UserBean customer;
23:
24:     @XmlElement(name="items")
25:     CartItemsWrapper items;
26:
27:     @XmlElement
28:     String total;
29:
30:     @XmlElement
31:     String shipping;
32:
33:     @XmlElement
34:     String HST;
35:
36:     @XmlElement
37:     String grandTotal;
38:
39:     public PurchaseOrderWrapper() {};
40:
41:     public PurchaseOrderWrapper(CartModel cart, int orderId) {
42:         this.id = orderId;
43:         this.submitted = new Date();
44:         this.customer = cart.getAccount();
45:         this.total = cart.getTotal().toString();
46:         this.shipping = cart.getShipping().toString();
47:         this.HST = cart.getTax().toString();
48:         this.grandTotal = cart.getGrandTotal().toString();
49:         this.items = new CartItemsWrapper(cart.getCartItems());
50:     }
51:     /**
52:      * @return the grandTotal
53:      */
54:     public String getGrandTotal() {
55:         return grandTotal;
56:     }
57:
58:     /**
59:      * @return the total
60:      */
61:     public String getTotal() {
62:         return total;
63:     }
64:
65:     /**
66:      * @return the shipping
67:      */
68:     public String getShipping() {
69:         return shipping;
70:     }
```

```
71:
72:      /**
73:       * @return the hST
74:       */
75:      public String getHST() {
76:          return HST;
77:      }
78:
79:
80:
81:
82: }
83:
```

```
1: package model;
2:
3: import javax.xml.bind.annotation.XmlAttribute;
4: import javax.xml.bind.annotation.XmlElement;
5:
6: public class UserBean {
7:
8:     @XmlAttribute(name="account")
9:     String name;
10:
11:
12:     String fullName;
13:
14:
15:     boolean admin;
16:
17:     public UserBean() {
18:
19:     }
20:
21:     public UserBean(String name, String fullName, boolean admin) {
22:         super();
23:         this.name = name;
24:         this.fullName = fullName;
25:         this.admin = admin;
26:     }
27:
28:
29:
30:     /**
31:      * @return the name
32:      */
33:     public String getName() {
34:         return name;
35:     }
36:
37:     /**
38:      * @return the admin
39:      */
40:     public boolean isAdmin() {
41:         return admin;
42:     }
43:
44:
45:     /**
46:      * @return the fullName
47:      */
48:     @XmlElement(name="name")
49:     public String getFullName() {
50:         return fullName;
51:     }
52:
53:
54: }
```

```
1: package tags;
2:
3: import java.io.IOException;
4:
5: import javax.servlet.jsp.JspException;
6: import javax.servlet.jsp.JspWriter;
7: import javax.servlet.jsp.tagext.SimpleTagSupport;
8:
9: public class MinSec extends SimpleTagSupport
10: {
11:     private static int THOUSAND = 1000;
12:     private static int BASE = 60;
13:     private long milliseconds;
14:
15:     public MinSec()
16:     {
17:         super();
18:     }
19:
20:     @Override
21:     public void doTag() throws JspException, IOException
22:     {
23:         super.doTag();
24:
25:         long seconds = milliseconds / THOUSAND;
26:         long m = seconds / BASE;
27:         long s = seconds % BASE;
28:
29:         JspWriter out = this.getJspContext().getOut();
30:         if (m != 0)
31:             out.write(m + "m");
32:         out.write(s + "s");
33:     }
34:
35:     public void setMilliseconds(long milliseconds)
36:     {
37:         this.milliseconds = milliseconds;
38:     }
39: }
```

```
1: package util;
2:
3: import javax.servlet.http.HttpServletRequest;
4:
5:
6: public abstract class Authenticator {
7:
8:     protected String sessionKey;
9:
10:    public abstract boolean isAuthenticated(HttpServletRequest request);
11:
12:    public abstract boolean login(HttpServletRequest request, String username,
13:                                String password) throws Exception;
14:
15:    public abstract void logout(HttpServletRequest request) throws Exception;
16:
17:
18: }
```



```

1: package util;
2:
3: import java.sql.Connection;
4: import java.sql.SQLException;
5:
6: import javax.naming.Context;
7: import javax.naming.InitialContext;
8: import javax.naming.NamingException;
9: import javax.sql.DataSource;
10:
11: public class eFoodsDataSource {
12:
13:     private static final String DATASOURCE_CONTEXT = "java:/comp/env/jdbc/EECS
";
14:     private static final String SCHEMA = "roumani";
15:     private static DataSource dataSource;
16:
17:
18:     /**
19:      * @return the dataSource
20:      * @throws NamingException
21:      */
22:     public static DataSource getDataSource() throws NamingException {
23:         if (dataSource == null) {
24:             synchronized(eFoodsDataSource.class) {
25:                 if (dataSource == null) {
26:                     Context initialContext = new InitialContext
t();
27:                     dataSource = (DataSource) initialContext.l
ookup(DATASOURCE_CONTEXT);
28:                 }
29:             }
30:         }
31:         return dataSource;
32:     }
33:
34:     public static Connection getConnection() throws SQLException, NamingExcept
ion {
35:         Connection c = getDataSource().getConnection();
36:         c.createStatement().executeUpdate("set schema " + SCHEMA);
37:         return c;
38:     }
39:     protected eFoodsDataSource() {}
40:
41:
42: }

```

```

1: package util;
2:
3: import java.io.File;
4: import java.io.FileWriter;
5:
6: import java.util.ArrayList;
7: import java.util.Arrays;
8: import java.util.Date;
9: import java.util.List;
10: import java.util.regex.Matcher;
11: import java.util.regex.Pattern;
12:
13: public class PurchaseOrderFiles {
14:     public class PurchaseOrderFile {
15:         public static final String STATUS_NEW = "new";
16:         public static final String STATUS_PENDING = "pending";
17:         public static final String STATUS_PURCHASED = "purchased";
18:
19:         String fileName;
20:         int orderId;
21:         String userName;
22:         String status;
23:         String modifiedDate;
24:         /**
25:          * @return the modifiedDate
26:          */
27:         public String getModifiedDate() {
28:             return modifiedDate;
29:         }
30:
31:         /**
32:          * @return the fileName
33:          */
34:         public String getFileName() {
35:             return fileName;
36:         }
37:
38:         /**
39:          * @return the oderId
40:          */
41:         public int getOrderId() {
42:             return orderId;
43:         }
44:
45:         /**
46:          * @return the userName
47:          */
48:         public String getUserName() {
49:             return userName;
50:         }
51:
52:         /**
53:          * @return the status
54:          */
55:         public String getStatus() {
56:             return status;
57:         }
58:
59:         public String getFileNameOnly() {
60:             return new File(fileName).getName();
61:         }
62:
63:         /**
64:          * This constructor will automatically determine the file name giv
65:          * en the required information.
66:          * Ensures that file names with illiegal schemes will not be saved
67:          */
68:         public PurchaseOrderFile(File basePath, int orderId, String userNa
me,

```

```

68:         String status) throws Exception {
69:         String filename = orderId + "_" + userName + "_" + status
+ ".xml";
70:         if (!filename.matches(MATCHER_REGEX))
71:             throw new Exception("Order filename invalid. Check
parameters. " + filename);
72:         File leFile = new File(basePath, filename);
73:         this.fileName = leFile.getAbsolutePath();
74:         this.orderId = orderId;
75:         this.userName = userName;
76:         this.status = status;
77:         this.modifiedDate = new Date(leFile.lastModified()).toStri
ng();
78:     }
79: }
80: protected static final String MATCHER_REGEX = "^(?<poId>[1-9][0-9]*)_(?<ac
countId>[A-Za-z0-9]+)_(?<statusId>new|pending|purchased)\\.xml$";
81:
82:
83:
84: private File basePath;
85: private List<PurchaseOrderFile> purchaseOrderFiles;
86:
87: public PurchaseOrderFiles(String basePath) throws Exception {
88:     this.basePath = new File(basePath);
89:     this.purchaseOrderFiles = allOrderFiles();
90: }
91:
92: /*
93:  * Traverses all files in the purchase order folder and creates a List of
94:  * PurchaseOrderFile classes ready to be used by other methods.
95:  */
96: public List<PurchaseOrderFile> allOrderFiles() throws Exception {
97:     File[] files = basePath.listFiles();
98:     List<PurchaseOrderFile> results = new ArrayList<PurchaseOrderFile>
();
99:     List<Integer> idArray = new ArrayList<Integer>();
100:     for (File file : files) {
101:         if (file.isFile()) {
102:             if (!file.getName().endsWith(".xml")) continue; //
Avoid stupid mac osx .DS_store files
103:             Matcher matcher = Pattern.compile(MATCHER_REGEX).m
atcher(file.getName());
104:             if (matcher.matches()) {
105:                 int currId = Integer.parseInt(matcher.grou
p("poId"));
106:                 String account = matcher.group("accountId"
);
107:                 String status = matcher.group("statusId");
108:                 if (idArray.contains(currId))
109:                     throw new Exception("Non unique id
" + currId );
110:                 else
111:                     idArray.add(currId);
112:                 results.add(new PurchaseOrderFile(basePath
,currId,account,status));
113:             } else {
114:                 throw new Exception("Malformed xml name "
+ file.getName());
115:             }
116:         }
117:     }
118:     return results;
119: }
120:
121: /*
122:  * Finds the highest order id and return int added 1
123:  */
124: public int getNextOrderId() throws Exception {
125:     int[] ids = new int[purchaseOrderFiles.size()];

```

```
126:         int i = 0;
127:         for(PurchaseOrderFile pof: purchaseOrderFiles)
128:             ids[i] = pof.getId();
129:             i++;
130:         Arrays.sort(ids);
131:         if (ids.length > 0) {
132:             return ids[ids.length - 1] + 1;
133:         } else {
134:             return 1;
135:         }
136:     }
137:
138:     /*
139:      * Creates a new purchase order file
140:      */
141:     public void storeNewOrder(int orderId, String accountName, String order) throws Exception {
142:         int nextId = getNextOrderId();
143:         if (nextId != orderId)
144:             throw new Exception("Order ID mismatch. ");
145:         PurchaseOrderFile pof = new PurchaseOrderFile(basePath, orderId, accountName, PurchaseOrderFile.STATUS_NEW);
146:         FileWriter fWriter = null;
147:         try {
148:             fWriter = new FileWriter(pof.getFileName());
149:             fWriter.write(order);
150:             purchaseOrderFiles.add(pof);
151:             System.out.println("PO: Stored purchase order " + pof.getFileName());
152:         } catch (Exception e) {
153:             throw e;
154:         } finally {
155:             if (fWriter != null)
156:                 fWriter.close();
157:         }
158:     }
159:
160:
161:     public List<PurchaseOrderFile> getOrdersByUser(String username) {
162:         List<PurchaseOrderFile> result = new ArrayList<PurchaseOrderFile>(
163: );
164:         for(PurchaseOrderFile pof : purchaseOrderFiles) {
165:             if (pof.getUserName().equals(username))
166:                 result.add(pof);
167:         }
168:         return result;
169:     }
170:
171:     public List<PurchaseOrderFile> getOrdersByStatus(String status) {
172:         List<PurchaseOrderFile> result = new ArrayList<PurchaseOrderFile>(
173: );
174:         for(PurchaseOrderFile pof : purchaseOrderFiles) {
175:             if (pof.getStatus().equals(status))
176:                 result.add(pof);
177:         }
178:         return result;
179:     }
180:
181:     public PurchaseOrderFile getOrderById(int orderId) {
182:         for(PurchaseOrderFile pof : purchaseOrderFiles) {
183:             if (pof.getId() == orderId)
184:                 return pof;
185:         }
186:         return null;
187:     }
188:
189:     /*
190:      * Renames an order file in order to change it's status
191:      */
192:     public void updateStatus(int orderId, String status) throws Exception {
```

```
191:         PurchaseOrderFile pof = getOrderById(orderId);
192:         if (pof != null) {
193:             String fname = pof.getFileName();
194:             PurchaseOrderFile newPof = new PurchaseOrderFile(basePath,
orderId, pof.getUserName(), status);
195:             String newFname = newPof.getFileName();
196:             File oldFile = new File(fname);
197:             File newFile = new File(newFname);
198:             if (oldFile.renameTo(newFile)) {
199:                 this.purchaseOrderFiles = allOrderFiles();
200:             } else {
201:                 throw new Exception("Could not rename file");
202:             }
203:         } else {
204:             throw new Exception("Order ID does not exist");
205:         }
206:
207:     }
208:
209:     /*
210:      * Deletes all order files in the system. Destructive action.
211:      */
212:     public void nuke() throws Exception {
213:         for(PurchaseOrderFile pof : purchaseOrderFiles) {
214:             new File(pof.getFileName()).delete();
215:         }
216:         this.purchaseOrderFiles = allOrderFiles();
217:     }
218:
219:
220: }
```

```
1: package util;
2:
3: import java.util.regex.*;
4:
5: import javax.servlet.http.HttpServletRequest;
6:
7: public class Route {
8:
9:     public static final String METHOD_GET = "GET";
10:    public static final String METHOD_POST = "POST";
11:    public static final String METHOD_ANY = "ANYTHING";
12:
13:    private String urlPattern;
14:    private String destination;
15:    private String method;
16:    private int destinationType;
17:    private boolean requireAuthentication;
18:    private Matcher matcher;
19:    private int identifier;
20:    private boolean requireAdmin;
21:
22:    /**
23:     * @return the matcher
24:     */
25:    public Matcher getMatcher() {
26:        return matcher;
27:    }
28:
29:    public Route(String urlPattern, String destination, String method, int identifier,
30:        boolean requireAuthentication) {
31:        this(urlPattern, destination, method, identifier, requireAuthentication, false);
32:    }
33:
34:    public Route(String urlPattern, String destination, String method, int identifier,
35:        boolean requireAuthentication, boolean requireAdmin) {
36:        super();
37:        this.urlPattern = urlPattern;
38:        this.destination = destination;
39:        this.method = method;
40:        this.identifier = identifier;
41:        this.requireAuthentication = requireAuthentication;
42:        this.requireAdmin = requireAdmin;
43:    }
44:
45:    /**
46:     * @return the identifier
47:     */
48:    public int getIdentifier() {
49:        return identifier;
50:    }
51:
52:    /**
53:     * @param identifier the identifier to set
54:     */
55:    public void setIdentifier(int identifier) {
56:        this.identifier = identifier;
57:    }
58:
59:    /**
60:     * @return the urlPattern
61:     */
62:    public String getUrlPattern() {
63:        return urlPattern;
64:    }
65:
66:    /**
67:     * @return the destination
```

```

68:         */
69:         public String getDestination() {
70:             return destination;
71:         }
72:
73:         /**
74:          * @return the destinationType
75:          */
76:         public int getDestinationType() {
77:             return destinationType;
78:         }
79:
80:         /**
81:          * @return the method
82:          */
83:         public String getMethod() {
84:             return method;
85:         }
86:
87:         /**
88:          * @return the requireAuthentication
89:          */
90:         public boolean isRequireAuthentication() {
91:             return requireAuthentication;
92:         }
93:
94:         /**
95:          * @return the requireAdmin
96:          */
97:         public boolean isRequireAdmin() {
98:             return requireAdmin;
99:         }
100:
101:         /**
102:          * @param requireAdmin the requireAdmin to set
103:          */
104:         public void setRequireAdmin(boolean requireAdmin) {
105:             this.requireAdmin = requireAdmin;
106:         }
107:
108:         /**
109:          * Tests if the given http request matches to this route
110:          */
111:         public boolean match(HttpServletRequest request) {
112:             Matcher m = Pattern.compile(this.urlPattern).matcher(request.getPa
thInfo());
113:             if (m.matches() && verifyMethod(request.getMethod())) {
114:                 this.matcher = m;
115:                 return true;
116:             } else {
117:                 return false;
118:             }
119:         }
120:
121:         private boolean verifyMethod(String m) {
122:             if (m == null || this.method.equals(METHOD_ANY)
|| this.method.equals(m))
123:                 return true;
124:             else
125:                 return false;
126:         }
127:
128:     }
129:
130: }

```

```
1: package util;
2:
3: import java.util.ArrayList;
4: import java.util.List;
5:
6: import javax.servlet.http.HttpServletRequest;
7:
8: public class Router {
9:
10:     public static final String REQUEST_ROUTE_KEY = "request-route";
11:
12:     List<Route> routes;
13:
14:     public Router() {
15:         this.routes = new ArrayList<Route>();
16:     }
17:
18:     public void addRoute(Route route) {
19:         routes.add(route);
20:     }
21:
22:     /**
23:      * Loop through all configured routes and find one that matches.
24:      * @param request
25:      * @return the route with respect to the request
26:      */
27:     public Route getRoute(HttpServletRequest request) {
28:         for (Route r : routes)
29:             if (r.match(request)){
30:                 request.setAttribute(REQUEST_ROUTE_KEY, r);
31:                 return r;
32:             }
33:
34:         return null;
35:     }
36:
37:     /**
38:      * @return the routes
39:      */
40:     public List<Route> getRoutes() {
41:         return routes;
42:     }
43:
44: }
```



```

1: package util;
2:
3: import java.net.URLDecoder;
4: import java.security.NoSuchAlgorithmException;
5: import java.security.SecureRandom;
6: import java.util.Enumeration;
7: import java.util.HashMap;
8: import java.util.List;
9: import java.util.Map;
10: import java.util.Random;
11:
12: import javax.crypto.Mac;
13: import javax.crypto.spec.SecretKeySpec;
14: import javax.servlet.http.HttpServletRequest;
15: import javax.servlet.http.HttpServletResponse;
16: import javax.servlet.http.HttpSession;
17: import javax.xml.bind.DatatypeConverter;
18:
19: import org.apache.catalina.util.RequestUtil;
20: import org.apache.tomcat.util.codec.binary.Base64;
21:
22: import model.UserBean;
23:
24: public class SSOAuthenticator extends Authenticator {
25:
26:     private static final String SESSION_KEY_PREFIX = "com.eFoods.util.Authenti
cation.SSO";
27:     private static final String NONCE_VALUE_KEY = "com.eFoods.util.Authenticat
ion.SSO.nonce.value";
28:     private static final String NONCE_TIME_KEY = "com.eFoods.util.Authenticati
on.SSO.nonce.time";
29:     private static final int NONCE_EXPIRY = 10; //minutes
30:     private static final int NONCE_LENGTH = 16;
31:
32:     private String sharedKey;
33:     private String sessionKey;
34:     private String ssoEndpoint;
35:     private String ssoReciever;
36:     private List<String> adminUsers;
37:
38:     public SSOAuthenticator(String ssoEndpoint, String ssoReciever, String sh
aredKey, List<String> adminUsers) throws Exception {
39:         if (ssoEndpoint == null || ssoEndpoint.length() < 1)
40:             throw new Exception("Endpoint must not be blank!");
41:         if (sharedKey == null || sharedKey.length() < 1)
42:             throw new Exception("Shared key must not be blank!");
43:         if (adminUsers.size() < 1)
44:             throw new Exception("No admin users in the system!");
45:         if (ssoReciever == null || ssoReciever.length() < 1)
46:             throw new Exception("SSO Recieve URL must not be empty!");
47:         this.ssoEndpoint = ssoEndpoint;
48:         this.ssoReciever = ssoReciever;
49:         this.sharedKey = sharedKey;
50:
51:         //Creates a unique sesion key for this SSO endpoint
52:         this.sessionKey = SESSION_KEY_PREFIX + "." + ssoEndpoint.hashCode(
);
53:         this.adminUsers = adminUsers;
54:
55:     }
56:
57:     @Override
58:     public boolean isAuthenticated(HttpServletRequest request) {
59:         UserBean user = getUser(request);
60:         Route currentRoute = (Route) request.getAttribute(Router.REQUEST_R
OUTE_KEY);
61:         if (user != null && currentRoute != null) {
62:             boolean needAdmin = currentRoute.isRequireAdmin();
63:             boolean hasAdmin = user.isAdmin();
64:             return needAdmin ? needAdmin == hasAdmin : true;

```

```

65:         }
66:         return false;
67:     }
68:
69:     public UserBean getUser(HttpServletRequest request) {
70:         return ((UserBean) request.getSession().getAttribute(this.sessionKey));
71:     }
72:
73:     private void setUser(HttpServletRequest request, UserBean user) {
74:         request.getSession().setAttribute(this.sessionKey, user);
75:     }
76:
77:     /**
78:      * Generates a nonce, constructs a payload and redirects to the auth endpoint
79:      */
80:     public void SSORedirect(HttpServletRequest request, HttpServletResponse response) throws Exception {
81:         String nonce = generateNonce(NONCE_LENGTH);
82:         String rawPayload = "nonce=" + nonce;
83:         String payload = new String(Base64.encodeBase64(rawPayload.getBytes()));
84:         String signature = encode(sharedKey, payload).toLowerCase();
85:
86:         String encodedPayload = URLEncoder.encode(payload, "UTF-8");
87:         String encodedSignature = URLEncoder.encode(signature, "UTF-8");
88:         String encodedReceiver = URLEncoder.encode(ssoReceiver, "UTF-8");
89:         String queryString = "payload=" + encodedPayload + "&signature=" + encodedSignature + "&redirect=" + encodedReceiver;
90:         storeNonce(request, nonce);
91:         System.out.println("Auth: Nonce created.");
92:         response.sendRedirect(ssoEndpoint + "?" + queryString);
93:     }
94:
95:     /**
96:      * Accepts a payload and a signature and verifies the authenticity of the request by comparing the nonce.
97:      * Set's the user in to the current session.
98:      */
99:     @Override
100:    public boolean login(HttpServletRequest request, String uname, String pwd) throws Exception {
101:        Map<String, String[]> payload = new HashMap<String, String[]>();
102:        String localSignature, localNonce, remoteNonce, remotePayload, remoteSignature, rawPayload, userName, userFullName;
103:
104:        remotePayload = request.getParameter("payload");
105:        remoteSignature = request.getParameter("signature");
106:        localSignature = encode(sharedKey, remotePayload);
107:        localNonce = retrieveNonce(request);
108:
109:        //Only use the nonce once!
110:        expireNonce(request);
111:
112:        rawPayload = new String(Base64.decodeBase64(remotePayload));
113:        RequestUtil.parseParameters(payload, rawPayload, "UTF-8");
114:
115:        System.out.println("Auth : remote Payload: " + remotePayload);
116:        System.out.println("Auth : raw Payload: " + rawPayload);
117:
118:        try {
119:            remoteNonce = payload.get("nonce")[0];
120:            userName = payload.get("username")[0];
121:            userFullName = payload.get("fullname")[0];
122:        } catch (Exception e) {
123:            throw new Exception("Invalid payload.");
124:        }
125:    }

```

```

128:
129:
130:         //Initial validation
131:         if (!localSignature.toLowerCase().equals(remoteSignature.toLowerCa
se()))
132:             throw new Exception("Payload verification failed.");
133:         if (localNonce == null)
134:             throw new Exception("Request expired. Please re-initiate l
ogin.");
135:         if (!localNonce.equals(remoteNonce))
136:             throw new Exception("Invalid nonce. Please re-initiaite lo
gin.");
137:         //End initial validation
138:
139:         //Data validation
140:         if (userName == null || userName.length() < 1)
141:             throw new Exception("Invalid username data in payload");
142:         if (userFullName == null || userFullName.length() < 1)
143:             throw new Exception("Invalid user full name data in payloa
d");
144:
145:         /*
146:         * FOLLOWING CODE IS COMMENTED OUT FOR THE SAKE OF MARKER. AS THER
E IS NO WAY
147:         * TO ADD HIM/HER AS AN ADMIN WITHOUT PRIOR KNOWLEDGE OF THEIR USE
R NAMES
148:         */
149:         //boolean isAdmin = false;
150:         //if (adminUsers.contains(userName))
151:         //    isAdmin = true;
152:         boolean isAdmin = true;
153:         /*
154:         * End testing code
155:         */
156:
157:         setUser(request, new UserBean(userName, userFullName, isAdmin));
158:         return isAuthenticated(request);
159:     }
160:
161:     @Override
162:     public void logout(HttpServletRequest request) throws Exception {
163:         HttpSession session = request.getSession(false);
164:         //Clears the session before invalidating
165:         if (session != null) {
166:             Enumeration<String> e = session.getAttributeNames();
167:             while (e.hasMoreElements()) {
168:                 String key = (String) e.nextElement();
169:                 session.setAttribute(key, null);
170:             }
171:             session.invalidate();
172:         }
173:         if (isAuthenticated(request))
174:             throw new Exception("Could not log out successfully!");
175:     }
176:
177:     protected static void storeNonce(HttpServletRequest request, String nonce)
{
178:         request.getSession().setAttribute(NONCE_VALUE_KEY, nonce);
179:         request.getSession().setAttribute(NONCE_TIME_KEY, System.currentTi
meMillis());
180:     }
181:
182:     protected static String retrieveNonce(HttpServletRequest request) {
183:         Long timeAtStore;
184:         try {
185:             timeAtStore = (Long) request.getSession().getAttribute(NON
CE_TIME_KEY);
186:         } catch (NumberFormatException e) {
187:             timeAtStore = null;
188:         }

```

```

189:         if (timeAtStore != null && timeAtStore > (System.currentTimeMillis
() - (NONCE_EXPIRY * 1000 * 60 * 10))) {
190:             return (String) request.getSession().getAttribute(NONCE_VA
LUE_KEY);
191:         } else {
192:             //Expire the nonce if the maximum time has been passed.
193:             expireNonce(request);
194:             return null;
195:         }
196:
197:
198:     }
199:
200:     protected static void expireNonce(HttpServletRequest request) {
201:         request.getSession().setAttribute(NONCE_VALUE_KEY, null);
202:         request.getSession().setAttribute(NONCE_TIME_KEY, null);
203:         System.out.println("Auth: Nonce destroyed");
204:     }
205:     protected static String encode(String secret, String data) throws Exceptio
n {
206:         String hashFunction = "HmacSHA256";
207:         Mac instance = Mac.getInstance(hashFunction);
208:         SecretKeySpec keySpec = new SecretKeySpec(secret.getBytes(),
209:             hashFunction);
210:
211:         instance.init(keySpec);
212:         return DatatypeConverter.printHexBinary((instance.doFinal(data.get
Bytes()))).toLowerCase();
213:     }
214:
215:     protected static String generateNonce(int length) throws NoSuchAlgorithme
ception {
216:         byte[] nonce = new byte[NONCE_LENGTH];
217:         Random rand = SecureRandom.getInstance ("SHA1PRNG");
218:         rand.nextBytes(nonce);
219:         return DatatypeConverter.printHexBinary(nonce);
220:     }
221:
222:
223:
224: }

```

```
1: <?xml version="1.0" encoding="UTF-8" ?>
2: <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0">
3:     <jsp:directive.page contentType="text/html; charset=UTF-8"
4:         pageEncoding="UTF-8" session="false"/>
5:     <jsp:output omit-xml-declaration="true" />
6:
7:
8:     <div class="col-md-6">
9:         <div class="contact">
10:             <p> Contact Information </p>
11:             <p> 4700 Keele St, Toronto, ON M3J 1P3 </p>
12:             <p> (416) 736-2100 </p>
13:         </div>
14:
15:     </div>
16:
17:     <div class="col-md-6">
18:         <iframe width="600" height="450" frameborder="0" style="border:0"
19:             src="https://www.google.com/maps/embed/v1/place?q=York%20Universit
y%2C%20Keele%20Street%2C%20Toronto%2C%20ON%2C%20Canada&key=AIzaSyDqdeEV19s2GM_hCojleu
-pKy37wB5_U9I"></iframe>
20:     </div>
21:
22: </jsp:root>
```

```
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" version="2.0">
4:     <jsp:directive.page contentType="text/html; charset=UTF-8"
5:         pageEncoding="UTF-8" session="false" />
6:     <jsp:output omit-xml-declaration="true" />
7:
8:     <!-- Cart / Log In -->
9:     <div class="col-md-12 header">
10:         <!-- Title -->
11:         <div class="pull-left">
12:             <div class="logo">
13:                 <h1><a href="/eFoods/">eFoods</a></h1>
14:             </div>
15:         </div>
16:         <div class="pull-right">
17:             <div class="cart">
18:                 <div class="btn-group">
19:                     <button type="button" class="btn btn-defau
lt btn-lg" data-ajaxify="backend/user" id="user-info">
20:
21:                         </button>
22:
23:                     <button type="button" class="btn btn-defau
lt btn-lg" data-ajaxify="backend/cart/badge" id="cart-info">
24:
25:                         </button>
26:                     </div>
27:                 </div>
28:             </div>
29:
30:         </div>
31: </jsp:root>
```

```

1: <?xml version="1.0" encoding="UTF-8" ?>
2: <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0">
3:     <jsp:directive.page contentType="text/html; charset=UTF-8"
4:         pageEncoding="UTF-8" session="false" />
5:     <jsp:output doctype-root-element="html"
6:         doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
7:         doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitiona
l.dtd"
8:         omit-xml-declaration="true" />
9:     <html xmlns="http://www.w3.org/1999/xhtml">
10: <head>
11: <meta name="viewport" content="width=device-width, initial-scale=1"></meta>
12: <meta charset="utf-8"></meta>
13: <title>Foods R Us</title>
14:
15: <!-- Start CSS -->
16: <!-- Latest compiled and minified CSS -->
17: <link rel="stylesheet"
18:     href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.1/css/bootstrap.min.cs
s"
19:     type="text/css" />
20: <link href="//maxcdn.bootstrapcdn.com/font-awesome/4.2.0/css/font-awesome.
min.css" rel="stylesheet"
21:     type="text/css"/>
22: <link href="//cdnjs.cloudflare.com/ajax/libs/bootstrap-material-design/0.1
.6/css/material-wfont.css" rel="stylesheet"
23:     type="text/css"/>
24: <link href="//cdnjs.cloudflare.com/ajax/libs/bootstrap-material-design/0.1
.6/css/ripples.min.css" rel="stylesheet"
25:     type="text/css"/>
26: <link href="//cdnjs.cloudflare.com/ajax/libs/bootstrap-material-design/0.1
.6/fonts/Material-Design-Icons.woff" rel="stylesheet"
27:     type="text/css"/>
28: <link href='http://fonts.googleapis.com/css?family=Lato:300,400' rel='styl
esheet'
29:     type='text/css'/>
30: <link rel="stylesheet"
31:     href="${pageContext.request.contextPath}/res/css/spinkit.css"
32:     type="text/css"/>
33: <link rel="stylesheet"
34:     href="${pageContext.request.contextPath}/res/css/main.css"
35:     type="text/css" title="cse4413" media="screen, print" />
36:
37: <!-- Start JS -->
38: </head>
39: <body>
40:     <!-- alert container -->
41:     <div id="alert-container"><jsp:text /></div>
42:     <!-- wrapper over content -->
43:     <div class="container">
44:
45:         <!-- header -->
46:         <div class="row">
47:             <jsp:include page="Head.jspx" />
48:         </div>
49:
50:         <div class="jumbotron">
51:             <jsp:include page="/partials/_search-bar.jspx">
52:                 </jsp:include>
53:             </div>
54:
55:
56:             <div class="row" data-ajaxify="backend/category/" id="main-content
">
57:                 <p class="lead text-center"> Please wait loading... </p>
58:             </div>
59:
60:
61:             <!-- footer -->
62:             <div class="row footer">

```

```
63:                                <jsp:include page="Footer.jspx" />
64:                                </div>
65:        </div>
66:
67:        <script type="text/javascript"
68:            src="${pageContext.request.contextPath}/res/js/main.js"><jsp:text
/></script>
69:        <script type="html/partial" id="alert-partial">
70: <div class="efoods-alert alert-hide" onclick="eFoods.util.hideAlert(this)">
71:     <div class="panel panel-danger">
72:         <div class="panel-heading">
73:             <h3 class="panel-title">Warning</h3>
74:         </div>
75:         <div class="panel-body">
76:             ###MSG###
77:         </div>
78:     </div>
79: </div>
80:                                </script>
81: </body>
82: </html>
83: </jsp:root>
```



```

1: <?xml version="1.0" encoding="UTF-8"?>
2: <Context privileged="true" reloadable="true">
3:   <WatchedResource>WEB-INF/web.xml</WatchedResource>
4:   <Manager pathname=""/>
5:   <Resource name="jdbc/EECS" factory="org.apache.tomcat.jdbc.pool.DataSourceFactor
y"
6:     type="javax.sql.DataSource"
7:     driverClassName="org.apache.derby.jdbc.ClientDriver"
8:     url="jdbc:derby://roumani.eecs.yorku.ca:64413/CSE"
9:     username="student" password="secret"/>
10:   <ResourceLink global="jdbc/EECS" name="jdbc/EECS" type="javax.sql.DataSource"/>
11: </Context>

```

```

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:my="urn:jsptld:/WEB-INF/taglib.tld"
5:     version="2.0">
6:     <jsp:directive.page contentType="text/html; charset=UTF-8"
7:         pageEncoding="UTF-8" session="false" />
8:     <jsp:output omit-xml-declaration="true" />
9:
10:
11:     <h1>Foods R Us! Data Analytics</h1>
12:     <br/>
13:     <table id="tb" border="0">
14:         <tr>
15:             <td><strong>Average time for customers to add an item to c
art:</strong></td>
16:             <td><my:MinSec milliseconds="\${avgAddItemTime}"/></td>
17:         </tr>
18:         <tr>
19:             <td><strong>Average time between a fresh visit to check ou
t:</strong></td>
20:             <td><my:MinSec milliseconds="\${avgCheckOutTime}"/></td>
21:         </tr>
22:     </table>
23:
24: </jsp:root>

```

```

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" version="2.0">
4:     <jsp:directive.page contentType="text/html; charset=UTF-8"
5:         pageEncoding="UTF-8" session="false" />
6:     <jsp:output omit-xml-declaration="true" />
7:     <a href="backend/category/" data-refresh-ids="main-content" onClick="return eFoods
.app.handleHref(this, true);">Back to all categories</a>
8:     <table class="table table-striped table-hover">
9:         <thead>
10:             <tr>
11:                 <th>Item</th>
12:                 <th>Price</th>
13:                 <th>Quantity</th>
14:                 <th>Total</th>
15:             </tr>
16:         </thead>
17:         <tbody>
18:             <c:forEach items="${cartItems}" var="r">
19:                 <tr>
20:                     <td>
21:                         <a href="backend/item/${r.item.number}" d
ata-refresh-ids="main-content" onClick="return eFoods.app.handleHref(this, true);">${r.it
em.name }</a>
22:                     </td>
23:                     <td>${r.item.price }</td>
24:                     <td>
25:                         <form method="POST" action="backend/cart"
26:                             data-refresh-ids="cart-info, main-content"
27:                             onsubmit="return eFoods.app.handleForm(thi
s);">
28:                             <input type="number" name="qty" value="${r
.quantity }" /> <input type="hidden"
29:                                 name="item" value="${r.item.number
}" /> <input type="submit"
30:                                     class="" name="itemAdd" value="Cha
nge" />
31:                         </form>
32:                     </td>
33:                     <td>${r.getItemTotal() }</td>
34:                 </tr>
35:             </c:forEach>
36:         </tbody>
37:     </table>
38:
39:     <div class="checkout-pricing">
40:         <p>Total: ${total.toPlainString() }</p>
41:         <p>Shipping: ${shipping.toPlainString() }</p>
42:         <p>Tax: ${tax.toPlainString() }</p>
43:         <p>SubTotal: ${total.add(shipping).add(tax).toPlainString() }</p>
44:     </div>
45:
46:
47:     <c:choose>
48:         <c:when test="${not empty user}">
49:             <form method="POST" action="backend/cart/checkout" >
50:                 <input type="submit" class="btn btn-primary btn-lg" name="
cartSubmit" value="Checkout" />
51:             </form>
52:         </c:when>
53:         <c:otherwise>
54:             <a class="btn btn-primary btn-lg" href="backend/login" onClick="return eFo
ods.util.statefulHref(this);">Checkout</a>
55:         </c:otherwise>
56:     </c:choose>
57: </jsp:root>

```

```
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" version="2.0">
4:     <jsp:directive.page contentType="text/html; charset=UTF-8"
5:       pageEncoding="UTF-8" session="false" />
6:     <jsp:output omit-xml-declaration="true" />
7:
8:     <!-- TODO: ajax item counter -->
9:     <a href="backend/cart/" data-refresh-ids="main-content" onClick="return eF
oods.app.handleHref(this, true);">
10:         <span class="badge"> ${cartItemCount} </span>
11:         <span class="glyphicon glyphicon-shopping-cart"></span>
12:     </a>
13: </jsp:root>
```

```

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" version="2.0">
4:     <jsp:directive.page contentType="text/html; charset=UTF-8"
5:         pageEncoding="UTF-8" session="false" />
6:     <jsp:output omit-xml-declaration="true" />
7:
8:     <a href="backend/category/" data-refresh-ids="main-content" onClick="return eFoods
.app.handleHref(this, true);">Back to all categories</a>
9:
10:    <c:if test="${not empty alertInfo }">
11:        <div class="alert alert-dismissable alert-info">
12:            <strong>Heads up! </strong> ${alertInfo }
13:        </div>
14:    </c:if>
15:
16:    <c:if test="${not empty alertWarn }">
17:        <div class="alert alert-dismissable alert-warn">
18:            <strong>Warning! </strong> ${alertWarn }
19:        </div>
20:    </c:if>
21:
22:
23:
24:    <h1> Your purchase history. </h1>
25:    <h3> New Orders</h3>
26:        <table class="table table-striped table-hover">
27:            <thead>
28:                <tr>
29:                    <th>Order Id</th>
30:                    <th>Date Placed</th>
31:                    <th>View</th>
32:                </tr>
33:            </thead>
34:            <tbody>
35:                <c:forEach items="${newOrders }" var="r">
36:                    <tr>
37:                        <td>
38:                            ${r.orderId }
39:                        </td>
40:                        <td>${r.modifiedDate }</td>
41:                        <td>
42:                            <a href="PurchaseOrders/${r.getFileNameOnly() }">O
pen.</a>
43:                        </td>
44:                    </tr>
45:                </c:forEach>
46:            </tbody>
47:        </table>
48:
49:    <h3> Pending </h3>
50:        <table class="table table-striped table-hover">
51:            <thead>
52:                <tr>
53:                    <th>Order Id</th>
54:                    <th>Date Placed</th>
55:                    <th>View</th>
56:                </tr>
57:            </thead>
58:            <tbody>
59:                <c:forEach items="${pendingOrders }" var="r">
60:                    <tr>
61:                        <td>
62:                            ${r.orderId }
63:                        </td>
64:                        <td>${r.modifiedDate }</td>
65:                        <td>
66:                            <a href="PurchaseOrders/${r.getFileNameOnly() }">O
pen.</a>
67:                        </td>

```

```
68:                                     </tr>
69:                                 </c:forEach>
70:                             </tbody>
71:                         </table>
72:
73:
74: <h3> Purchased </h3>
75:     <table class="table table-striped table-hover">
76:         <thead>
77:             <tr>
78:                 <th>Order Id</th>
79:                 <th>Date Placed</th>
80:                 <th>View</th>
81:             </tr>
82:         </thead>
83:         <tbody>
84:             <c:forEach items="${purchasedOrders}" var="r">
85:                 <tr>
86:                     <td>
87:                         ${r.orderId }
88:                     </td>
89:                     <td>${r.modifiedDate }</td>
90:                     <td>
91:                         <a href="PurchaseOrders/${r.getFileNameOnly()} ">O
pen.</a>
92:                     </td>
93:                 </tr>
94:             </c:forEach>
95:         </tbody>
96:     </table>
97:
98: </jsp:root>
```

```
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" version="2.0">
4:           <jsp:directive.page contentType="text/html; charset=UTF-8"
5:           pageEncoding="UTF-8" session="false" />
6:           <jsp:output omit-xml-declaration="true" />
7:
8:           <h1>Congratulations!</h1>
9:           <p>Your order was successfully placed. </p>
10:          <p>You can check the status of all your orders by visiting your <a href="backend/c
art/history" data-refresh-ids="main-content" onClick="return eFoods.app.handleHref(this,
true);">purchase history</a> page</p>
11: </jsp:root>
```

```

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" version="2.0">
4:     <jsp:directive.page contentType="text/html; charset=UTF-8"
5:         pageEncoding="UTF-8" session="false" />
6:     <jsp:output omit-xml-declaration="true" />
7:
8:     <!-- TODO: add jsp looping over contents -->
9:     <!-- main content -->
10:    <!-- need to add a loop here to go through the content -->
11:    <!-- fields are
12:    picture
13:    name
14:    price
15:    item id
16:    -->
17:    <div class="col-md-12">
18:        <h1> All Categories </h1>
19:    </div>
20:    <c:forEach items="${results}" var="r">
21:        <div class="col-md-4 col-lg-3 col-sm-6">
22:            <div class="thumbnail well lg-well">
23:                <a href="backend/category/${r.id}" data-refresh-i
ds="main-content" onClick="return eFoods.app.handleHref(this, true);"> </img>
26:                </a>
27:                <div class="caption">
28:                    <p>${r.name }</p>
29:                    <p>${r.description }</p>
30:                    <p class="hidden">${r.id }</p>
31:                </div>
32:            </div>
33:        </div>
34:    </c:forEach>
35:
36: </jsp:root>

```



```

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" version="2.0">
4:     <jsp:directive.page contentType="text/html; charset=UTF-8"
5:         pageEncoding="UTF-8" session="false" />
6:     <jsp:output omit-xml-declaration="true" />
7:
8:
9:     <div class="thumbnail item well well-lg">
10:         <div class="caption">
11:             <p class="item-name">${item.name }</p>
12:             <p>\${item.price } CDN</p>
13:
14:
15:             <form method="POST"
16:                 data-refresh-ids="cart-info, item-${item.number }"
17:                 onsubmit="return eFoods.app.handleForm(this);" cla
18:                 ss="clearfix form-control-wrapper">
19:                 <c:choose>
20:                     <c:when test="${not empty cartItem }">
21:                         <span class="pull-left">
22:                             <label for="number">Qty:</label>
23:                             <input class="item-quantity form-control"
24:                                 type="number" name="qty" value="${cartItem.quantity }" /> <input type="hidden"
25:                                 name="item" value="${item.number }" />
26:                         </span>
27:                         <span class="pull-right">
28:                             <input type="submit"
29:                                 class="btn btn-primary" name="itemAdd" val
30:                                 ue="Change" />
31:                         </span>
32:                         <span class="pull-left"><small><b>Note:</b> This i
33:                         tem is in your cart.</small></span>
34:                     </c:when>
35:                     <c:otherwise>
36:                         <span class="pull-left">
37:                             <label for="number">Qty:</label>
38:                             <input class="item-quantity form-c
39:                                 ontrol" type="number" name="qty" value="1" />
40:                             <input type="hidden" name="item" v
41:                                 alue="${item.number }" />
42:                         </span>
43:                         <span class="pull-right">
44:                             <input type="submit" class="btn bt
45:                                 n-primary" name="itemAdd" value="Add" />
46:                         </span>
47:                     </c:otherwise>
48:                 </c:choose>
49:             </form>
50:         </div>
51:         <hr/>
52:         <div class="caption clearfix">
53:             <p class="h5" style="padding-bottom: 10px;"><b>Sug
54:             gested Purchase</b></p>
55:             <a href="backend/item/${crossSellItem.number }" da
56:                 ta-refresh-ids="main-content" onClick="return eFoods.app.handleHref(this, true);">
57:                 </img>
59:                 <span class="pull-right" style="max-width:
60:                 130px;">
61:                     <p>${crossSellItem.name }</p>
62:                     <p>\${crossSellItem.price } CDN</p>
63:                 </span>
64:             </a>
65:         </div>
66:     </div>
67:
68: </jsp:root>

```

```

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" version="2.0">
4:     <jsp:directive.page contentType="text/html; charset=UTF-8"
5:         pageEncoding="UTF-8" session="false" />
6:     <jsp:output omit-xml-declaration="true" />
7:
8:     <h1>${category.name }</h1>
9:     <a href="backend/item/filter/" data-refresh-ids="main-content"
10:         onClick="return eFoods.app.handleHref(this, true);">Back to all
11:         categories</a>
12:     <c:forEach items="${results }" var="r">
13:         <div class="col-md-4" data-ajaxify="backend/item/${r.number }" id=
"item-${r.number }">
14:             <c:set var="item" scope="request" value="${r }" />
15:             <c:set var="cartItem" scope="request" value="${cartItems[r
.name] }" />
16:             <jsp:include page="_item.jspx" />
17:         </div>
18:     </c:forEach>
19:
20: </jsp:root>

```

```

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" version="2.0">
4:     <jsp:directive.page contentType="text/html; charset=UTF-8"
5:         pageEncoding="UTF-8" session="false" />
6:     <jsp:output omit-xml-declaration="true" />
7:
8:
9:     <div class="thumbnail item well well-lg">
10:         </im
g>
11:         <div class="caption">
12:             <p class="item-name">${item.name }</p>
13:             <p>\${item.price } CDN</p>
14:
15:             <form method="POST" action="backen
d/cart"
16:                 data-refresh-ids="cart-info, item-${item.number }"
17:                 onsubmit="return eFoods.app.handleForm(this);" cla
ss="clearfix form-control-wrapper">
18:                 <c:choose>
19:                     <c:when test="${not empty cartItem }">
20:                         <span class="pull-left">
21:                             <label for="number">Qty:</label>
22:                             <input class="item-quantity form-control"
type="number" name="qty" value="${cartItem.quantity }" /> <input type="hidden"
23:                                 name="item" value="${item.number }" />
24:                         </span>
25:                         <span class="pull-right">
26:                             <input type="submit"
27:                                 class="btn btn-primary" name="itemAdd" val
ue="Change" />
28:                         </span>
29:                         <span class="pull-left"><small><b>Note:</b> This i
tem is in your cart.</small></span>
30:                     </c:when>
31:                     <c:otherwise>
32:                         <span class="pull-left">
33:                             <label for="number">Qty:</label>
34:                             <input class="item-quantity form-c
ontrol" type="number" name="qty" value="1" />
35:                             <input type="hidden" name="item" v
alue="${item.number }" />
36:                         </span>
37:                         <span class="pull-right">
38:                             <input type="submit" class="btn bt
n-primary" name="itemAdd" value="Add" />
39:                         </span>
40:                     </c:otherwise>
41:                 </c:choose>
42:             </form>
43:         </div>
44:     </div>
45:
46:
47: </jsp:root>

```

```

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" version="2.0">
4:     <jsp:directive.page contentType="text/html; charset=UTF-8"
5:         pageEncoding="UTF-8" session="false" />
6:     <jsp:output omit-xml-declaration="true" />
7:
8:     <div class="col-md-12">
9:         <p class="h4"><a href="backend/category" data-refresh-ids="main-co
ncontent" onClick="return eFoods.app.handleHref(this, true);">All Categories</a></p>
10:         <h1><a href="backend/category/${category.id}" data-refresh-ids="m
ain-content" onClick="return eFoods.app.handleHref(this, true);">${category.name }</a></h
1>
11:         <div class="icon-previous">
12:             <a href="javascript:window.history.back()"><span class="md
i-navigation-arrow-back"/></a>
13:         </div>
14:     </div>
15:
16:     <c:forEach items="${results}" var="r">
17:         <div class="col-md-4 col-sm-6 col-lg-3" data-ajaxify="backend/item
/${r.number }/partial" id="item-${r.number }">
18:             <c:set var="item" scope="request" value="${r }" />
19:             <c:set var="cartItem" scope="request" value="${cartItems[r
.name] }" />
20:             <jsp:include page="_item.jspx" />
21:         </div>
22:     </c:forEach>
23:
24: </jsp:root>

```

```

1: <?xml version="1.0" encoding="UTF-8" ?>
2: <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0">
3:     <jsp:directive.page contentType="text/html; charset=UTF-8"
4:         pageEncoding="UTF-8" session="false"/>
5:     <jsp:output doctype-root-element="html"
6:         doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
7:         doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional
l.dtd"
8:         omit-xml-declaration="true" />
9: <html xmlns="http://www.w3.org/1999/xhtml">
10: <head>
11: </head>
12: <body>
13:
14:     <div class="text-center row">
15:         <h2 class="home-heading"> The best way to shop and order Food.</h2
>
16:         <p> Choose foods ranging from meats to ice cream.</p>
17:         <div class="home-search-bar col-md-12">
18:             <form method="get" action="backend/search" id="search-form
"
19:                 data-refresh-ids="main-content" onSubmit="return false;">
20:                 <input name="filter" class="search-input" id="ite
m-search" type="text"
21:                     value="" placeholder="Search from a wide assortmen
t of foods."/>
22:                 <input type="hidden" name="page" value="1"/>
23:                 <input type="hidden" name="limit" value="25"/>
24:             </form>
25:         </div>
26:     </div>
27:
28: </body>
29: </html>
30: </jsp:root>

```

```
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" version="2.0">
4:     <jsp:directive.page contentType="text/html; charset=UTF-8"
5:       pageEncoding="UTF-8" session="false" />
6:     <jsp:output omit-xml-declaration="true" />
7:     <a href="javascript:window.history.back()">Go back.</a>
8:
9:     <h1>Looks like we got a problem...</h1>
10:    <br/>
11:    <c:if test="${not empty errorMessage}">
12:      <h2>${errorMessage}</h2>
13:    </c:if>
14:    <c:if test="${empty errorMessage}">
15:      <h2>We screwed up. Our bad. Please try again!</h2>
16:    </c:if>
17:
18:    <br/>
19:    <h3>Here's a picture of a kitten to brighten your day.</h3>
20:    <br/>
21:    </img>
22:  </jsp:root>
```

```
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" version="2.0">
4:   <jsp:directive.page contentType="text/html; charset=UTF-8"
5:     pageEncoding="UTF-8" session="false" />
6:   <jsp:output omit-xml-declaration="true" />
7:
8:   <c:choose>
9:     <c:when test="{not empty currentUser }">
10:      <p class="login-text">Hello, ${currentUser.fullName }! |<
/p>
11:      <c:if test="{currentUser.admin }">
12:        <p class="login-text">(<a href="backend/analytics"
data-refresh-ids="main-content" onClick="return eFoods.app.handleHref(this, true);">Anal
ytics</a>) |</p>
13:      </c:if>
14:      <p class="login-text"><a href="backend/cart/history" data-
refresh-ids="main-content" onClick="return eFoods.app.handleHref(this, true);">Purchase H
istory</a> |</p>
15:      <p class="login-text"><a href="backend/logout" data-refres
h-ids="user-info, cart-info, main-content" onClick="return eFoods.app.handleHref(this);">
Log Out</a></p>
16:    </c:when>
17:    <c:otherwise>
18:      <p class="login-text"><a href="backend/login" onClick="ret
urn eFoods.util.statefulHref(this);">Log In</a></p>
19:    </c:otherwise>
20:  </c:choose>
21:
22: </jsp:root>
```

```

1: <?xml version="1.0" encoding="UTF-8"?>
2: <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3:     <xsl:template match="/">
4:         <html>
5:             <head>
6:                 <title>Purchase Order Report</title>
7:             </head>
8:             <body>
9:                 <h1> Order Information </h1>
10:                <xsl:apply-templates/>
11:            </body>
12:        </html>
13:    </xsl:template>
14:
15:    <xsl:template match="order">
16:        <div>
17:            <p>ID: <xsl:value-of select="@id"/></p>
18:            <p>Time Submitted: <xsl:value-of select="@submitted"/></p>
19:        </div>
20:
21:        <xsl:apply-templates select="customer"/>
22:        <table border="1" cellpadding="5">
23:            <tr>
24:                <th>Item Number</th>
25:                <th>Item Name</th>
26:                <th>Price</th>
27:                <th>Quantity</th>
28:                <th>Extended</th>
29:            </tr>
30:            <xsl:for-each select="./items/item">
31:                <tr>
32:                    <td><xsl:value-of select="@number"/></td>
33:                    <td><xsl:value-of select="./name"/></td>
34:                    <td><xsl:value-of select="./price"/></td>
35:                    <td><xsl:value-of select="./quantity"/></td>
36:                    <td><xsl:value-of select="./extended"/></td>
37:                </tr>
38:            </xsl:for-each>
39:        </table>
40:
41:        <h3> Costs </h3>
42:        <p>Total: <xsl:value-of select="./total"/></p>
43:        <p>Shipping: <xsl:value-of select="./shipping"/></p>
44:        <p>HST: <xsl:value-of select="./HST"/></p>
45:        <p>Grand Total: <xsl:value-of select="./grandTotal"/></p>
46:    </xsl:template>
47:
48:    <xsl:template match="customer">
49:        <p>Customer Name: <xsl:value-of select="./name"/></p>
50:    </xsl:template>
51:
52: </xsl:stylesheet>

```



```

1: /*
2:  * Main CSS rules
3:  */
4:
5: html, body {
6:     margin-top: 20px;
7:     font-family: 'Lato';
8:     font-size: 14px;
9: }
10:
11: .header {
12:     margin-bottom: 100px;
13: }
14:
15: .login-text {
16:     display: inline;
17:     font-size: 1.0em;
18: }
19:
20: .footer {
21:     margin-top: 100px;
22: }
23:
24: .logo {
25:
26: }
27:
28: a {
29:     text-decoration: none;
30:     color: inherit;
31: }
32:
33: a:hover {
34:     text-decoration: none;
35: }
36:
37: /* Level this with the logo */
38: .cart {
39:     padding-top: 20px;
40: }
41:
42: .home-landing {
43: }
44:
45: .home-heading {
46:     font-family: Lato;
47:     font-weight: 300;
48: }
49:
50: .home-search-bar {
51:     padding-left: 20px;
52:     margin-top: 50px;
53:     margin-bottom: 50px;
54: }
55:
56: .search-input {
57:     height: 2em;
58:     font-size: 2em;
59:     width: 100%;
60:     padding-left: 60px;
61: }
62:
63: .contact > p {
64:     font-size: 1.45em;
65: }
66:
67: .item-quantity {
68:     width: 30px;
69:     margin: 15px 5px;
70:     display: inline-block;

```

```

71:         text-align: center;
72:     }
73:
74:     .contact {
75:         background: white;
76:         padding: 10px;
77:         height: 450px /* Make sure this matches the map */
78:     }
79:
80:     .item {
81:         border-radius: 5px;
82:         margin: 10px;
83:         padding: 10px !important;
84:         height: 335px;
85:     }
86:
87:     .icon-previous {
88:         font-size: 60px;
89:         display: inline-block;
90:         background: white;
91:         border-radius: 20px;
92:     }
93:
94:     .checkout-pricing {
95:         font-size: 1.5em;
96:     }
97:
98:     .item p {
99:         margin: 0;
100:        padding: 0;
101:    }
102:
103:    .item .item-name {
104:        text-overflow: ellipsis;
105:        white-space: nowrap;
106:        overflow: hidden;
107:    }
108:
109:    .efoods-alert
110:    {
111:        position: fixed;
112:        margin-right: auto;
113:        margin-left: auto;
114:        margin-top: 20px;
115:        width: 500px;
116:        left: 0;
117:        right: 0;
118:        z-index: 9999;
119:    }
120:    .efoods-alert:hover {
121:        cursor: pointer;
122:    }
123:    .efoods-alert.alert-show {
124:        animation: fadein 0.3s;
125:        -moz-animation: fadein 0.3s; /* Firefox */
126:        -webkit-animation: fadein 0.3s; /* Safari and Chrome */
127:        -o-animation: fadein 0.3s; /* Opera */
128:    }
129:
130:    .efoods-alert.alert-hide {
131:        animation: fadeout 0.3s;
132:        -moz-animation: fadeout 0.3s; /* Firefox */
133:        -webkit-animation: fadeout 0.3s; /* Safari and Chrome */
134:        -o-animation: fadeout 0.3s; /* Opera */
135:    }
136:
137:    @keyframes fadein {
138:        from {
139:            opacity: 0;
140:        }

```

```
141:     to {
142:         opacity:1;
143:     }
144: }
145: @-moz-keyframes fadein { /* Firefox */
146:     from {
147:         opacity:0;
148:     }
149:     to {
150:         opacity:1;
151:     }
152: }
153: @-webkit-keyframes fadein { /* Safari and Chrome */
154:     from {
155:         opacity:0;
156:     }
157:     to {
158:         opacity:1;
159:     }
160: }
161: @-o-keyframes fadein { /* Opera */
162:     from {
163:         opacity:0;
164:     }
165:     to {
166:         opacity: 1;
167:     }
168: }
169:
170: @keyframes fadeout {
171:     from {
172:         opacity:1;
173:     }
174:     to {
175:         opacity:0;
176:     }
177: }
178: @-moz-keyframes fadeout { /* Firefox */
179:     from {
180:         opacity:1;
181:     }
182:     to {
183:         opacity:0;
184:     }
185: }
186: @-webkit-keyframes fadeout { /* Safari and Chrome */
187:     from {
188:         opacity:1;
189:     }
190:     to {
191:         opacity:0;
192:     }
193: }
194: @-o-keyframes fadeout { /* Opera */
195:     from {
196:         opacity:1;
197:     }
198:     to {
199:         opacity:0;
200:     }
201: }
```

```
1: /**
2:  *
3:  */
4:
5: //For archaic browsers that doesn't support console logging (*cough* IE.. *cough)
6: if (typeof console == "undefined") {
7:     this.console = {log: function() {}};
8: }
9:
10:
11: //Define the app scope
12: var eFoods = {};
13: //Utility functions
14: eFoods.util = {};
15: //Standard errors -- todo:
16: eFoods.util.errors = {};
17: //Core app code
18: eFoods.app = {};
19: //App vars
20: eFoods.vars = {
21:     MAIN_URL : "backend/category",
22:     CONTENT_DIV : "main-content"
23: };
24:
25: eFoods.util.removeClass = function(element, className) {
26:     element.className = element.className.replace(new RegExp(className, 'g'), '');
27: }
28: eFoods.util.addClass = function(element, className) {
29:     element.className += " " + className;
30: }
31: eFoods.util.showAlert = function(msg) {
32:     var partial = document.getElementById("alert-partial");
33:     var partialHTML = partial.innerHTML.replace('###MSG###', msg);
34:     var alertContainer = document.getElementById("alert-container");
35:     alertContainer.innerHTML = partialHTML;
36:     var alerts = alertContainer.querySelectorAll(".efoods-alert");
37:     for(var i=0;i < alerts.length; i++) {
38:         eFoods.util.removeClass(alerts[i], "alert-hide");
39:         eFoods.util.addClass(alerts[i], "alert-show");
40:         setTimeout(function(element) {
41:             return function() {
42:                 eFoods.util.hideAlert(element);
43:             }
44:         })(alerts[i], 5000);
45:     }
46: }
47:
48: eFoods.util.hideAlert = function(element) {
49:     console.log(element);
50:     var alertContainer = document.getElementById("alert-container");
51:     eFoods.util.removeClass(element, "alert-show");
52:     eFoods.util.addClass(element, "alert-hide");
53:     setTimeout(function(el) {
54:         return function() {
55:             alertContainer.removeChild(el);
56:         }
57:     })(element), 300);
58: }
59:
60: // Returns the baseURL, ex. http://localhost:4413
61: eFoods.util.baseURL = function() {
62:     path = window.location.href.split('/');
63:     protocol = path[0];
64:     host = path[2];
65:     return protocol + "://" + host;
66: }
67:
68: //Ajax helper. Takes url and 2 callbacks.
69: eFoods.util.doAjax = function(url, data, method, onSuccess, onFailure) {
70:
```

```
71:         //Default callbacks when not supplied
72:         if (!onSuccess) {
73:             onSuccess = function(request) {
74:                 console.log("Request successful");
75:                 console.log(request);
76:             }
77:         }
78:         if (!onFailure) {
79:             onFailure = function(request) {
80:                 console.log("Request failed");
81:                 console.log(request);
82:                 try {
83:                     var msg = JSON.parse(request.responseText);
84:                     eFoods.util.showAlert(msg.error);
85:                 } catch (e) {
86:                     console.log(request.responseText)
87:                 }
88:             }
89:         }
90:         //End default callbacks
91:
92:         //Handler for xmlhttprequest
93:         var handler = function(request) {
94:             if ((request.readyState == 4)) {
95:                 if (request.status == 200) {
96:                     onSuccess(request);
97:                 } else {
98:                     onFailure(request)
99:                 }
100:             }
101:         }
102:     }
103:
104:     var request = new XMLHttpRequest();
105:     var postData = null;
106:     var hasQS = url.split("?").length > 1;
107:     url = url + (hasQS ? "&" : "?") + "ajax=true";
108:
109:     if (method.trim().toUpperCase() == "GET") { //If GET we append data with a
'?' mark
110:         url += data.length > 0 ? "&" + data : ""
111:         request.open(method, url, true);
112:     } else if (method.trim().toUpperCase() == "POST") { //If POST we set the c
ontent type and assign data as postData
113:         postData = data;
114:         request.open(method, url, true);
115:         request.setRequestHeader('Content-type',
116:             'application/x-www-form-urlencoded');
117:     } else {
118:         console.log(this, method);
119:         throw "dont know about that http method";
120:     }
121:
122:     request.onreadystatechange = function() {
123:         handler(request);
124:     };
125:     request.send(postData);
126: };
127:
128: //Convert values in a form in to a query string for ajax submission
129: eFoods.util.formToQueryString = function(form) {
130:     var elem = form.elements;
131:     var params = "";
132:     for (var i = 0; i < elem.length; i++) {
133:         if (elem[i].tagName == "SELECT") {
134:             value = elem[i].options[elem[i].selectedIndex].value;
135:         } else {
136:             value = elem[i].value;
137:         }
138:         params += elem[i].name + "=" + encodeURIComponent(value) + "&"
```

```

139:         }
140:         return params;
141:     }
142:
143: //Get current app state from hashbang
144: eFoods.util.getState = function() {
145:     var state = "";
146:     if (window.location.hash && window.location.hash.substring(0, 2) == "#!")
{
147:         state = window.location.hash.substring(2);
148:     }
149:     return state.length < 1 ? null : state
150: }
151:
152: //Set hashbang state (or default state when empty) to app state
153: eFoods.util.setState = function() {
154:     var state = this.getState();
155:     if (state) {
156:         url = state
157:     } else {
158:         url = eFoods.vars.MAIN_URL;
159:     }
160:     var ele = document.getElementById(eFoods.vars.CONTENT_DIV);
161:     ele.setAttribute('data-ajaxify', url)
162: }
163:
164: //Ajaxify, looks for the tag data-ajaxify and populates the inside via ajax usisn
g "GET"
165: eFoods.app.ajaxify = function(element) {
166:     var url = element.getAttribute("data-ajaxify");
167:     var isMain = false;
168:     // Check if we're changing app state
169:     if (element.getAttribute("id") == eFoods.vars.CONTENT_DIV) {
170:         window.location.hash = "#!" + url;
171:         isMain = true;
172:     }
173:
174:     eFoods.util.doAjax(url, (isMain ? "main=true" : ""), "GET", function(reque
st) {
175:         console.log("Ajaxified.", url, element.id);
176:         element.innerHTML = request.responseText;
177:     });
178: };
179:
180: //Ajaxify everything with the tag!
181: eFoods.app.ajaxifyAll = function() {
182:     var elements = document.querySelectorAll('[data-ajaxify]');
183:     for (i = 0; i < elements.length; i++) {
184:         (function(target) {
185:             eFoods.app.ajaxify(target)
186:         })(elements[i]); //This a little bit of js sorcery called self-inv
oking closure scoping.
187:     }
188: }
189:
190: //Invoke this to call a hyperlink. See the dom element for the required tags. This
will also refresh given dom elements on success (used to refresh cart badge and user bad
ge)
191: //ajaxifyTarget = on true, instead of refreshing the target, perform the ajax call
and put the response in to the target dom element (used to change pages - category, item
, car)
192: eFoods.app.handleHref = function(href, ajaxifyTarget) {
193:     try {
194:         var url = href.getAttribute("href");
195:         var refreshDoms = href.getAttribute("data-refresh-ids").split(",")
;
196:         if (ajaxifyTarget) {
197:             for (i = 0; i < refreshDoms.length; i++) {
198:                 (function(target) {
199:                     target.setAttribute("data-ajaxify", url);

```

```
200:         eFoods.app.ajaxify(target);
201:     })(document.getElementById(refreshDoms[i].trim()))
;
202:     }
203:     } else {
204:         eFoods.util.doAjax(url, "", "GET", function(request) {
205:             console.log("Request success!" + url);
206:             for (i = 0; i < refreshDoms.length; i++) {
207:                 (function(target) {
208:                     eFoods.app.ajaxify(target);
209:                 })(document.getElementById(refreshDoms[i].
trim()))
210:             }
211:         });
212:     }
213:
214:     } catch (e) {
215:         //throw e
216:         console.log(e);
217:     }
218:     return false;
219: };
220:
221: //Ajax form submission
222: eFoods.app.handleForm = function(form, ajaxifyTarget) {
223:     try {
224:         var method = form.method;
225:         var url = form.getAttribute('action');
226:         var refreshDoms = form.getAttribute("data-refresh-ids").split(",")
;
227:         var formData = eFoods.util.formToQueryString(form);
228:         console.log("Submitting form", method, url, refreshDoms, formData,
form);
229:
230:         if (ajaxifyTarget) {
231:             for (i = 0; i < refreshDoms.length; i++) {
232:                 (function(target) {
233:                     console.log("Target url for form" , url +
"?" + formData);
234:                     target.setAttribute("data-ajaxify", url +
"?" + formData);
235:                     eFoods.app.ajaxify(target);
236:                 })(document.getElementById(refreshDoms[i].trim()))
;
237:             }
238:         } else {
239:             eFoods.util.doAjax(url, formData, method, function(request
) {
240:                 console.log("Request success!" + url);
241:                 for (i = 0; i < refreshDoms.length; i++) {
242:                     (function(target) {
243:                         eFoods.app.ajaxify(target);
244:                     })(document.getElementById(refreshDoms[i].
trim()))
245:                 }
246:             });
247:         }
248:     } catch (e) {
249:         console.log(e, e.stack);
250:     }
251:     return false;
252: };
253:
254: //Watch for any changes in the hash bang, update app state if needed
255: eFoods.app.watchState = function() {
256:     var ele = document.getElementById(eFoods.vars.CONTENT_DIV);
257:     var interval = setInterval(function() {
258:         var appState = ele.getAttribute('data-ajaxify');
259:         if (appState != eFoods.util.getState()) {
260:             eFoods.util.setState();
```

```
261:             eFoods.app.ajaxify(ele);
262:         }
263:     }, 250)
264: }
265:
266: eFoods.util.statefulHref = function(ele) {
267:     var state = eFoods.util.getState();
268:     var hasQS = ele.href.split("?").length > 1;
269:     var toAppend = (hasQS ? "&" : "?") + (state ? "state=" + encodeURIComponent(state) : "");
270:     window.location.href = ele.href + toAppend;
271:     return false;
272: }
273:
274: eFoods.util.addEvents = function(handlers, ev, fn) {
275:     console.log('Adding event for', ev, handlers);
276:     for (var i = 0; i < handlers.length; i++) {
277:         var h = handlers[i];
278:         h.addEventListener(ev, fn);
279:     }
280: }
281:
282: //Initialization code. We'll call this at the bottom of the file.
283: eFoods.app.init = function() {
284:     console.log(eFoods.util.baseURL());
285:
286:     eFoods.util.setState();
287:     eFoods.app.ajaxifyAll();
288:     eFoods.app.watchState();
289:
290:
291: };
292:
293: document.onreadystatechange = function () {
294:     if (document.readyState == "interactive") {
295:         eFoods.app.init();
296:     }
297: }
298:
299: /* The idea for this module is as the user is typing
300: * the main page is updated with foods matching the search.
301: * 
302: * We use a timeout to avoid sending requests for every keystroke.
303: */
304: var searchBar = document.getElementById('item-search');
305: var searchForm = document.getElementById('search-form');
306:
307: var searchTimeout;
308: searchBar.addEventListener('keyup', function() {
309:     // an ajax request will be only sent only after 400ms
310:     if (searchTimeout) {
311:         clearTimeout(searchTimeout);
312:     }
313:     searchTimeout = setTimeout(function() {
314:         // do ajax here
315:         eFoods.app.handleForm(searchForm, true);
316:     }, 400)
317: })
```



```

1: <?xml version="1.0"?>
2: <!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty
/configure.dtd">
3: <Configure class="org.eclipse.jetty.webapp.WebAppContext">
4:     <New id="DS" class="org.eclipse.jetty.plus.jndi.Resource">
5:         <Arg></Arg>
6:         <Arg>java:comp/env/jdbc/EECS</Arg>
7:         <Arg>
8:             <New class="org.apache.derby.jdbc.ClientConnectionPoolData
Source">
9:                 <Set name="DatabaseName">CSE</Set>
10:                <Set name="ServerName">roumani.eecs.yorku.ca</Set>
11:                <Set name="PortNumber">64413</Set>
12:                <Set name="User">student</Set>
13:                <Set name="Password">secret</Set>
14:            </New>
15:        </Arg>
16:    </New>
17: </Configure>

```

```
1: <?xml version="1.0" encoding="UTF-8" ?>
2:
3: <taglib xmlns="http://java.sun.com/xml/ns/j2ee"
4:         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5:         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
6:         http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
7:         version="2.0">
8:
9:     <description>custom tags for eFood</description>
10:    <tlib-version>1.0</tlib-version>
11:    <short-name>CustomTagLibraryEfood</short-name>
12:
13:    <tag>
14:        <description>converts milliseconds to #m#s format</description>
15:        <name>MinSec</name>
16:        <tag-class>tags.MinSec</tag-class>
17:        <body-content>empty</body-content>
18:        <attribute>
19:            <name>milliseconds</name>
20:            <required>true</required>
21:            <rtexprvalue>true</rtexprvalue>
22:        </attribute>
23:    </tag>
24:
25: </taglib>
```

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns
.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xml
ns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
3:   <display-name>eFoods</display-name>
4:   <welcome-file-list>
5:     <welcome-file>Home.jsp</welcome-file>
6:   </welcome-file-list>
7:   <servlet>
8:     <servlet-name>Category</servlet-name>
9:     <servlet-class>ctrl.Category</servlet-class>
10:   </servlet>
11:   <servlet>
12:     <servlet-name>Item</servlet-name>
13:     <servlet-class>ctrl.Item</servlet-class>
14:   </servlet>
15:   <servlet>
16:     <servlet-name>Auth</servlet-name>
17:     <servlet-class>ctrl.Auth</servlet-class>
18:   </servlet>
19:   <servlet>
20:     <servlet-name>Cart</servlet-name>
21:     <servlet-class>ctrl.Cart</servlet-class>
22:   </servlet>
23:   <servlet>
24:     <servlet-name>Misc</servlet-name>
25:     <servlet-class>ctrl.Misc</servlet-class>
26:   </servlet>
27:   <servlet>
28:     <servlet-name>Orders</servlet-name>
29:     <servlet-class>ctrl.Orders</servlet-class>
30:   </servlet>
31:   <servlet>
32:     <servlet-name>Analytics</servlet-name>
33:     <servlet-class>ctrl.Analytics</servlet-class>
34:   </servlet>
35: </web-app>
```