E-Foods

An E-Commerce Website

Designed and Developed By

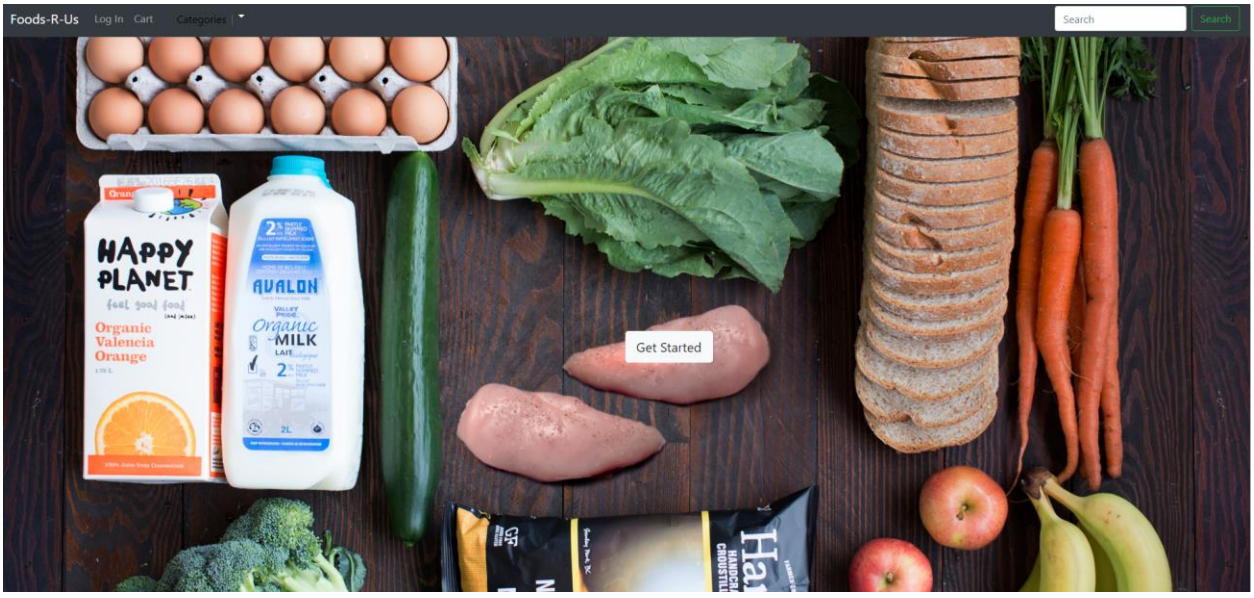
Amin Adam

Omair Anwar

Alexandra Zaslavsky

Submitted as the group project for the course EECS4413 - Building E-Commerce Systems

Prof. H. Roumani - Fall 2018



Acknowledgments

Much of the front end was implemented using Bootstrap classes using:

https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css

The image that was used for the landing page was from:

http://northernstyleexposure.com/wp-content/uploads/2017/03/weekly_staples_fromabove.jpg

Introduction

E-Foods is an ecommerce website that connects customers that require stock food orders and suppliers that wholesale food items. E-Foods does not stock any merchandise. The business model of E-Foods is order consolidation. Order consolidation means that E-Foods collects several orders from the clients, consolidates the orders into one single ship-able package. All orders may not be consolidated; so, the orders from clients are consolidated to minimize the amount of shipping required. It does this and that. It serves customers like this. To use this website customers, must do this and that. Bringing the complete multi-product order into one single package to be shipped saves on packaging, delivery and increases customer satisfaction.

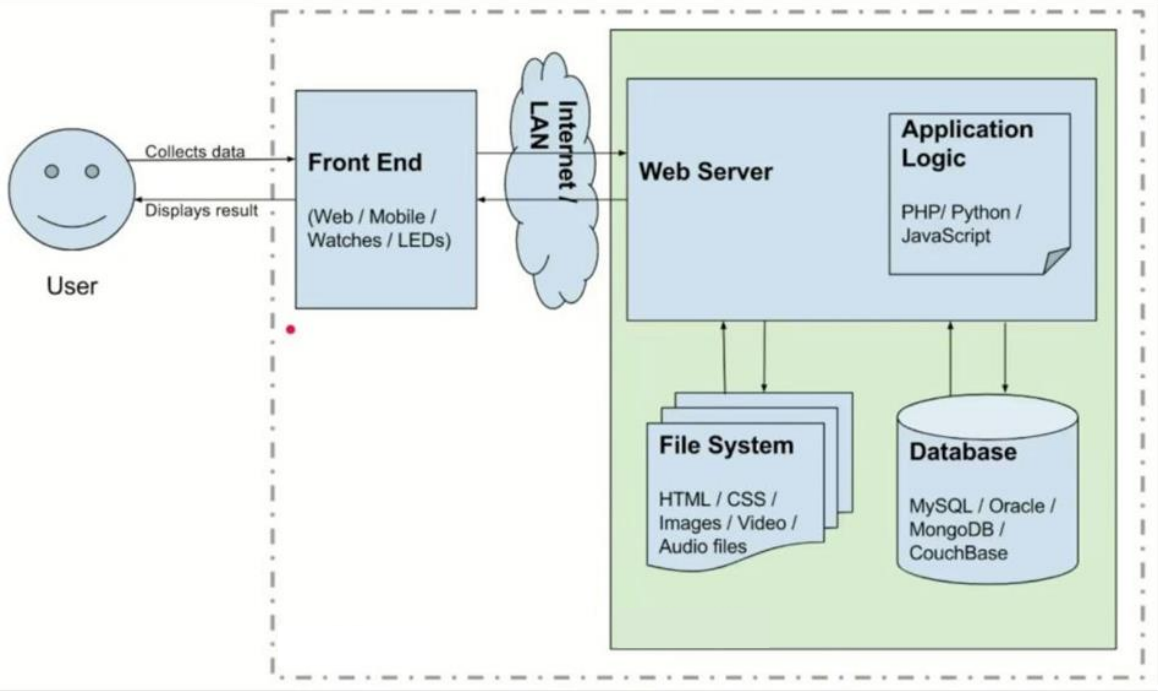
Design and Architecture

The design of the website was done taking into consideration several design considerations. Our design decisions are based on Reusability of code, Scalability of the services and Usability of the website. The primary consideration in the design of E-Foods website is Scalability. Content displayed on the website is dynamic and it can update on its own depending on what is present in the database. The administrators of the website do not have to code in the food items sold directly into the website. The items are directly taken from the database that is kept. The items in the database may change at any time and the website can present all available items to clients. It is scalable in that the website can handle increasing amount of food categories and items by dynamically retrieving the information from the database. Then the secondary concern was usability of the website. The website is designed to be user friendly and allows access to different parts of the website by having links available in the header that is present in all pages. Clients can access item categories using the dropdown option, they can use the versatile searching functionality to find items they are interested in. The searching feature is especially designed to simplify searching for items. It can take multiple search terms like names of products, product IDs and other information at once and display a list of items that match the search. Moreover, categories and items are presented in an appealing manner to allow customers to use the provided information make their purchase decisions. Thirdly, the website is designed to protect the identities of clients. The website

contains an authentication feature where any client that wants to purchase goods or observe their orders need to log in to the website through a secure third-party OAuth authentication service. Since keeping client information in a centralized database of E-Foods may expose clients to data breach attacks it was decided that a third-party authentication service that specializes in security and data protection is the best design choice. In addition, using third-party authentication allows the web site to be extensible in that payment services like PayPal or Cryptocurrency transaction services like Ethereum can be added to or removed from the website. Another significant design consideration is code reusability. Both the front end and the back end of the website is designed to be modular. Modular design allows for easy code reusability decreasing the amount of code required to handle services. For example, in the front end the header of the website persists through all the pages of the website. This was achieved by separating the header in a separate file and calling this module in every other page. Also items are displayed by navigating through categories or by searching using the search bar. Only one items page is used to display the contents since they are mutually exclusive content displays.

Architecture and Data Flow

The architecture of the website is a layered architecture. The website's entire architecture has three major layers. The top B2C layer where clients can browse products and make purchases, the MiddleWare layer that consolidates and produces procurement orders and the B2B layer that takes the procurement orders of the middleware layer and determines a supplier. The scope of the project is the top 2 layers. It is a layered architecture because the data flow of unidirectional from the top layer (B2B) to the middleware layer and finally to the B2B layer. The B2C layer is a multitier client-server inner architecture where several clients can access the services of the website. It has a web server that serves pages that contain the requested information. The business logic is within the server and it manages information retrieval from database, producing properly formatted output to the layer below (middleware). The Model-View-Controller design pattern is used to separate the concerns. The view simply presents data that is passed on to it from the controller and it provides clients an interactive GUI. The controller takes requests from the view and directs it to the appropriate handler in the model. The model takes data passed from the controller and depending on the data it retrieves information from the database or produces output files. One important feature of the website is authentication. Some pages in the website need proper credentials to access them. For example, ordering products requires users to log into their accounts. For this a secure third-party authentication is added. This authentication feature uses the OAuth authentication technology. It is separate from the other features in that it can be added or removed and improved at any time.



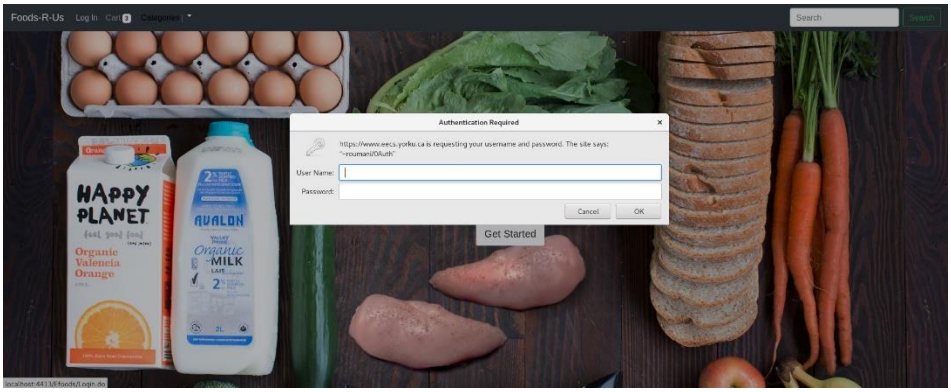
Implementation

This section discusses implementation of the major features of the website.

Styling and Content display

When we began to display content, it was unappealing. Our primary concern was to implement the functionalities. We wanted to make sure that all food categories are displayed in the dropdown list in the header, and in the categories page. We wanted to make sure that the correct items were retrieved from the database when a link for a category was clicked or when a user searches using the search bar. Once we made sure the functionalities were fulfilled, we used bootstrap CSS library and custom written CSS to style the pages. A design decision we made that was different from the requirements is that we displayed the categories as large cards with pictures and descriptive text and the items as medium sized cards where 3 items were arranged in a row. One difficulty we faced when styling was having a properly lining up the cards in the items page. The bootstrap font library and the custom written CSS were clashing so it made the arrangement less pleasing to the eyes. Even though the bugs took time to find they were solved.

Authentication



The requirements for the authentication is that it becomes a detachable service that can be implemented locally or outsourced to third parties like cloud OAuth providers. We decided to use the EECS OAuth authentication service provided by professor Roumani. A potential use case is as follows: user clicks the login button, a filter servlet intercepts the request and checks if the user is logged in; if the user is not logged in then it redirects the user to the EECS OAuth service along with a return link; if the user is already present in the database then authentication will be successful and the OAuth service redirects the page to the return address provided. One limitation with using the EECS OAuth service we found is that upon unsuccessful attempt it just takes the page to a generic page that contains error information instead of accepting another redirection link that redirects to a page we provide. In the beginning only one servlet was used to handle authentication requests from different pages. That means a user may log in from the homepage or when clicking checkout. Upon successful login the user is redirected to two different pages depending on where the authentication is called. Managing two different redirections in a single servlet complicated the code and as a result it was decided that the OAuth functionality should be handled by two filter servlets. One would detect the servlet path and the other would be called by the return link of the OAuth and it in turn redirects the user to the appropriate pages.

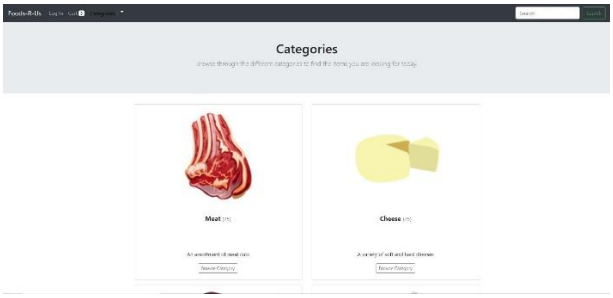
Browsing and Purchasing

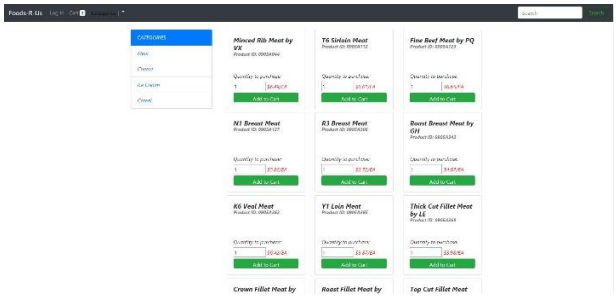
A user is able to browse the contents for sale by clicking the category button or selecting a category from the dropdown.

Clicking the category button sends a request to the Category.do servlet which returns with the categories and their images from the database. Upon clicking one of the categories, a user is redirected to the items page which holds items from the selected category.

Once a category is selected, all the items in that category are displayed as returned by the items.do servlet. This page also allows a user to navigate to a different category using the category bar on the left.

Alternatively, a user can browse items using the search feature. In this case all items that match search criteria are returned regardless of their category.

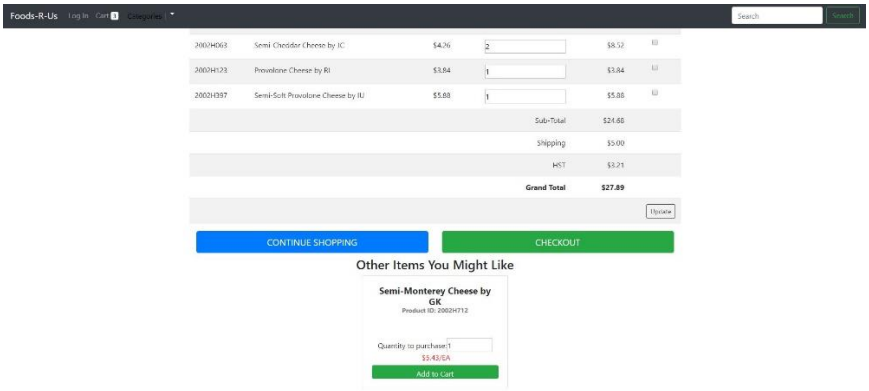




The above left image shows the categories page which the user can use to navigate the items and the items page on the right

Creating a Purchase Order

Once a user has checked out, a filter verifies that they have authenticated before the order can be created. If they have yet to authenticate, they are prompted to authenticate using the EECS OAuth implemented by Professor Roumani. The request is redirected to the ‘Order.do’ servlet.



The above image shows the checkout page. A recommendation is given to the customer based on their clicks and previous choices.

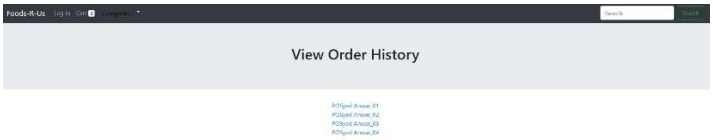
Here, we retrieve the user’s name and username as well as the orders all saved in the session. We verify that the order list is not empty and create an order using the Engine model’s create orders method. We verify that the correct folder path exists. If it doesn’t exist, we create the correct path on disk to save the placed orders on. We used the xml schema to create classes with xml binding to create all the necessary objects to populate such an order request. The parameters sent to this method were now parsed and the correct objects were created, looping through the list of items and setting all the necessary fields.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<orderType id="321" submitted="2018-11-27">
    <customer account="soanwar">
        <name>Syed Anwar</name>
    </customer>
    <items>
        <item number="0905A127">
            <name>N1 Breast Meat</name>
            <price>5.86</price>
            <quantity>1</quantity>
            <extended>5.86</extended>
        </item>
    </items>
    <total>5.86</total>
    <shipping>5.00</shipping>
    <HST>0.76</HST>
    <grandTotal>11.62</grandTotal>
</orderType>
```

The above image shows a sample PO created after a purchase. All necessary fields are filled and formatted. Purchase orders are created in the /home/user/PO/ directory. A sample PO listing would be /home/user/PO/Syed Anwar_02.xml (_Done after the 02 if it has been processed).

The items list attribute is then removed from the session and the user is redirected to a page to view their placed orders. If the use case that the cart was empty, the user is redirected without creating a placed order xml.

The order history



The user is then redirected to the orders page after a list of order names has been obtained from the disk. This page had links to the different orders that had been placed, while the ID code at the end represented the order number for that client.

Viewing an order

Order Details:

Order Number: 321

Order Date: 2018-11-27-05:00

Ordered Items:

Product Id	Name	Price	Quantity	Extended
1409S413	K5 Praline Ice Cream	6.44	1	6.44
2002H063	Semi-Cheddar Cheese by JC	4.26	2	8.52
2002H123	Provolone Cheese by RI	3.84	1	3.84
2002H397	Semi-Soft Provolone Cheese by IU	5.88	1	5.88

Sub-Total: 24.68

HST: 3.21

Shipping: 5.00

Grand Total: 32.89

Clicking one of these links would redirect to the show order servlet. This servlet obtains the file name and sends it to the engine's generateOrder() method, the result of which is and order set as a session attribute. An order is an OrderType object which is created by unmarshalling the correct xml file.

The Middleware

The middleware is a layer between the B2C and the B2B aspects of the business. It is implemented as a separate Java Project, which we can assume could be running on a different machine. This project uses the same object structure that was used to create the placed orders. It processes all orders that have not been processed yet, identified by the filename to create a procurement order for items from a large set of placed orders. It does this by first verifying that a destination folder exists and creating one if it doesn't. It reads the xml files and uses a procurement bean which has necessary fields to create an xml report. This report has one item appearing only once, with quantities, per item cost and extended cost. If an item exists in the list, its values are updated with the sum of its current value and the with the value of an item. HST is then calculated on the total amount which would be the taxes that the business pays.

On completion of this report, the placed orders are renamed by adding _Done to the filename to indicate that they've been processed. It is important to note that this report does not care who the user is when reading a placed order. Only information about items is processed and only reports that have not been processed are consulted

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<procurementBean id="13" submitted="2018-11-27">
    <name>our name</name>
    <items number="0905A044">
        <name>Minced Rib Meat by VX</name>
        <price>6.49</price>
        <quantity>3</quantity>
        <extended>19.47</extended>
    </items>
    <items number="2002H123">
        <name>Provolone Cheese by RI</name>
        <price>3.84</price>
        <quantity>2</quantity>
        <extended>7.68</extended>
    </items>
    <items number="1409S413">
        <name>K5 Praline Ice Cream</name>
        <price>6.44</price>
        <quantity>2</quantity>
        <extended>12.88</extended>
    </items>
    <items number="2002H712">
        <name>Semi-Monterey Cheese by GK</name>
        <price>5.43</price>
        <quantity>1</quantity>
        <extended>5.43</extended>
    </items>
    <total>45.27</total>
    <HST>6.26</HST>
    <grandTotal>51.53</grandTotal>
</procurementBean>
```

The image above on the right shows a sample procurement order created by processing multiple placed orders. Procurement orders are listed on the disk inside the middleware project under /home/user/ws_4413/eclipse-workspace/Middleware/POrders/. A sample listing would be /home/user/ws_4413/eclipse-workspace/Middleware/POrders/ProcurementOrder_07.xml.

The Team

Once the project was announced, we had an initial meeting to discuss the architecture of the site as well as the workflow. In this meeting, we determined the base classes we would need to create in order to have a solid foundation to work with. Amin was to create the DAO and java bean for the records in the ITEMS table, while Alex was to create the DAO and java bean for the records in the CATEGORY table. Meanwhile, Omair had begun to work on the purchase orders and B2C-B2B middleware. Afterwards, the division of work was less organized as when someone would complete the implementation of a feature they would begin working on the next feature which was determined to be most important. After the initial meeting, we worked independently, only meeting once a week to discuss potential design decisions.

While working on this project we faced many challenges. One of the challenges, we faced was source control. Since we worked independently, we each worked in different environments. As one of our members was using a windows machine, while others used linux the project committed to the github would be committed libraries specific to each operating system, resulting in the project not running on one system or the other. Eventually, we had come to the agreement that we would only use linux machines when working on the project and those that needed to, would install the virtualbox environment. Another lesson that was learned was how to work cooperatively while working independently. As each of had our own responsibilities for other courses, finding time to meet and work in person was very difficult. Since many features that we worked on had overlap with features others would work on, we had to find ways of communicating our implementations with enough detail so that others were able to work congruently.