

---

# CSE4413: BUILDING E-COMMERCE SYSTEMS

DECEMBER 5, 2013

PROJECT C: FOODS R US

THE SCRIPT KIDDIES, FALL 2013

---

HEBA ISHAQ (CSE73135), HERMAN SINGH BADWAL (SSJHERMA), HOSSEIN TORABI (CSE13239),  
MAYNIL PATEL (CSE93033) AND OSAMEDE EKE (CSE81021)

## TABLE OF CONTENTS

---

Table of Contents .....	2
Acknowledgements.....	5
Design .....	5
Data flow.....	5
Technology used.....	6
Database (Dao model) component.....	6
Search .....	7
Testing.....	7
Fragmented View.....	7
Web 3.0 .....	7
Controller Design .....	8
Best practice: Front Controller .....	8
Server Memory.....	8
Separation of Text used in View .....	8
Multi-threading considerations.....	8
Model Design.....	8
Development methodology.....	8
Source code management.....	9
View Design .....	9
Software Engineering .....	9
B2B component.....	9
B2B Design.....	9
Implementation.....	10
Controller Implementation .....	10
Front Controller.....	10
Server Memory.....	10
Separation of Text used in View .....	10

Multi-threading considerations.....	10
View Implementation issues and Separation of Concerns.....	11
CSS3 implementation .....	11
Four major categories were considered in establishing the CSS design for the entire website namely: .....	11
Model Implementation .....	11
B2C Security.....	12
SHOPPING Cart IMPLEMENTATION .....	12
Purchase Order Marshalling.....	12
Advertisement Filter .....	12
b2b considerations .....	12
The Team.....	13
Division of Work.....	13
meetings.....	13
Member contribution .....	14
How each member learned the other components.....	14
Output Samples.....	14
The Source Code.....	17
Appendix 1: Test cases for Behavior-driven development (use cases).....	18
Test Case #2: A client adds an item to the shopping cart .....	18
Test Case #3: A client logs in .....	18
Test Case #4: A client checks out .....	18
TEST Case #5: A client visits the URL of a P/O .....	18
Appendix B: Initial positions .....	18
Appendix c: B2C classes .....	20
<b>B2C View Delegation</b> .....	20
<b>B2C Client authentication</b> .....	20
<b>Cart</b> .....	21
<b>Purchase Order Marshaling</b> .....	21
<b>Advertisement Filter</b> .....	21
Appendix D: Original project timeline.....	21
Project Timeline .....	21

Prototyping phase.....	21
Iteration 1.....	21
Iteration 2.....	21
Iteration 3.....	22
Final Release phase .....	22
Appendix E: Detailed workflow .....	22
Appendix F: Enhancement backlog .....	22

## ACKNOWLEDGEMENTS

---

Special thanks to Professor Hamzeh Roumani for teaching us everything we know to make a website like this possible.

We'd also like to thank Abdullah Rubiyath for his advice and guidance in this Project.

Lastly, we'd like to acknowledge the following websites where we took CSS from:

<http://tympanus.net/codrops/2012/10/16/custom-login-form-styling/>

<http://tympanus.net/codrops/2011/11/02/original-hover-effects-with-css3/>

## DESIGN

---

The system architecture of our web application was designed with a multi-tier architecture and Model-View-Controller separation in mind. We endeavored to follow a 3-tier system which separates the presentation (JSPX, CSS, JS, Controller), business logic/processing (Models) and data management functions (DB, XML marshal/unmarshal) and further separated the presentation layer into the View and the Controller, consistent with both of these architectural patterns. This separation was done not only on the server side but also on the client side, with styling present in CSS (View), Business logic in JavaScript (similar to a Model) and the JSPX files containing the content (like a Controller). This separation of concerns makes our project more scalable, maintainable and understandable.

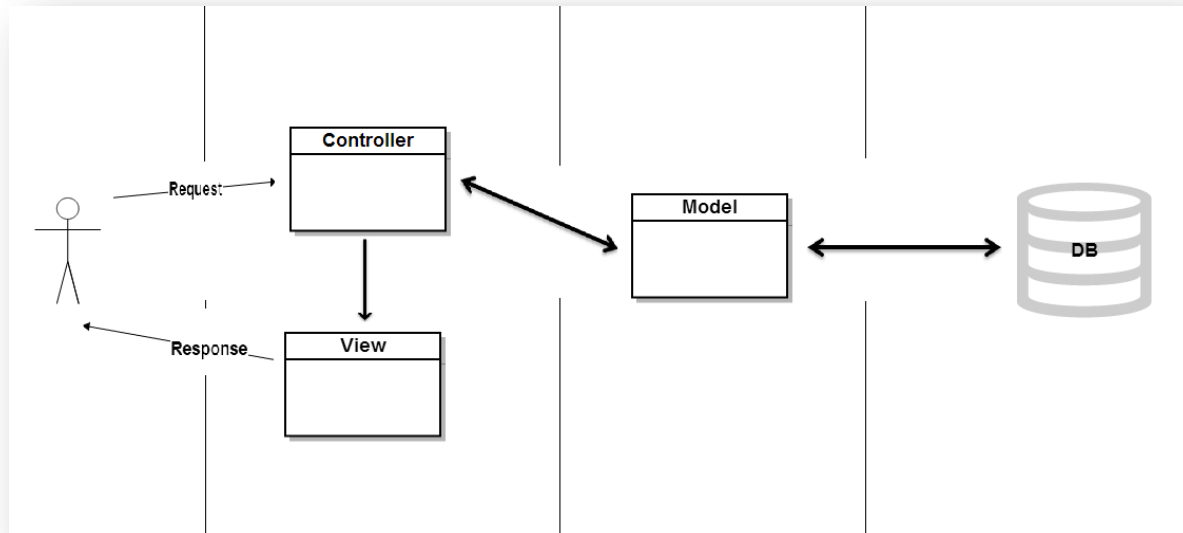
To demonstrate our commitment to separation of concerns, we originally had very amazing effects in the View taking full advantage of the CSS3 animations. However due to mixing concerns with JavaScript being present in the JSPX file and overly complicated CSS files taken from other websites, we decided to do an overhaul of the view in the interest of keeping it simple but easily understood, maintainable and extensible. In fact, there were three major UI versions: two major overhauls were performed over the course of this three week project (as will be discussed later).

A major effort was made to make the CSS understandable with a proper hierarchical structure requiring minimal change in order to maintain.

## DATA FLOW

---

Figure 1 illustrates the data flow between the different components. When a client makes a request it goes to the Filter/Listen in this case Dispatcher, checks if the request made is in valid, passes the request to the controller that delegates it to both model and view. The model which is responsible for the business logic requests the data from the DB through a class called DerbyDAO.java. Model then passes the data to the controller that passes it o the view where it is displayed to the user.



*Figure 1: High level data flow*

A more detailed workflow is available in Appendix E.

---

## TECHNOLOGY USED

- **Programming:** Java, SQL, JSPX, XSLT, XML (XSD) Schema.
- **Database:** JDBC, Derby
- **Client-Side:** HTML, CSS, JavaScript, AJAX
- **Frameworks:** J2EE, MVC
- **Application Servers:** Apache Tomcat Server, Servlets, Web Services
- **Testing:** EasyMock, JUnit
- **Source control:** GIT (BitBucket)
- **Tools:** Eclipse, IBM Smart Cloud, Gmail, Firebug, Google Doc, MS Office Suite, Whatsapp, Dropbox

---

## DATABASE (DAO MODEL) COMPONENT

We discussed at length the DAO component architecture. In large-scale deployments the Database interface component is separated into two sub-components: the DAL (Data Access Layer) and DAO (Data Access Object) parts. Each DAO would be specific to a database, for example different DAO classes for Oracle and DB2 access if they were both used. So each DAO would contain logic specific to that database. By contrast, the DAL class would provide a single interface for all the database access and make the appropriate function calls in the DAO required.

In this project, we were only using a single database: Derby. In addition, the database was read-only and there were only three (relatively small) tables to access. No joins were required for this project either. For this reason,

we decided that a single DAO class to access the database was enough. If the database access grows we may switch to having multiple DAOs in the future.

We also learned that the best way to query the database is to use prepared statements. We plan to implement this in the next phase of the project, but it is not currently ready due to time constraints. The benefit of prepared statements would be to provide parameters to “templates” that would only be compiled and optimized once. This helps with query optimization and makes database access quicker.

---

## SEARCH

---

We felt that the search option was an important feature for an eCommerce website. After all, if clients cannot find what they are looking for then how can we sell things? We had many things in the pipeline for search including a regular search field, having an express order page and providing an option to sort the items by name, price (both high to low and low to high) and perhaps options like product rating and so on in the future. We currently are able to display items by their category and the search is working well but in the implementation of the sorting feature we ran into problems. We could either do it by asking the database to do it but that would seem to require an Ajax call because it would be inconvenient to have a submit button in addition to a drop down list. Alternatively we could do the sorting with JavaScript on the client side which would probably be the better option and plan to do this in a future release. A more advanced search panel is also in the works where we could filter items by price and other factors.

---

## TESTING

---

A proper testing framework and ideology is one of the most important parts of a software development project as it ensures quality and gives an accurate picture of the progress. Test-driven and Behavior-Driven Development were both used in combination in our development lifecycle. Our team members had experience with both the JUnit and EasyMock unit testing frameworks. Black box testing was manual by trying out all the use cases given in the requirements document. You may see an extensive list of a checklist of test cases our testing lead used.

---

## FRAGMENTED VIEW

---

In order to easily maintain our website and make it feasible for future developments, we decided to use page fragments. This would simplify the process of building large and complex pages. Therefore, we broke down our pages into smaller page fragments using “*jsp:include*” tags. Based on the current architecture of the website we decided to use a *Header* fragment to be included in every page since it contains the most common features of the website including the main title, search box and login form.

---

## WEB 3.0

---

We wanted to design our website as much as possible to implement Web 3.0 features. In this way, a lot more emphasis would be put on the view side with advanced CSS, JavaScript and JSPX features dominating the web application. Ajax would be used to make calls to the servlet in this case. In a future release we still plan to use Ajax along with web technologies like JSON to transfer information.

---

## CONTROLLER DESIGN

---

---

### BEST PRACTICE: FRONT CONTROLLER

---

After learning about the Front controller concept in lecture we decided to break up our single Start controller up into pieces to ensure it was modular and easier to understand. This would make it a lot easier to extend and make changes as well. However due to the short timeline we were unable to make this change for this first release.

---

### SERVER MEMORY

---

Since web applications have many threads, there was an emphasis in the design to reduce memory usage and limit use of scopes to store information.

---

### SEPARATION OF TEXT USED IN VIEW

---

To make future translation of the website easier and to separate content from formatting, we decided to extract all strings used by the view into an XML file.

---

### MULTI-THREADING CONSIDERATIONS

---

Since we know that there are many threads in this web application, it was imperative to be aware of potential issues and mitigate them.

---

## MODEL DESIGN

---

Our model is broken down into multiple modules to maintain separation of concerns and to implement new feature as requirements changes along with time.

The focus in the B2C model was to ensure that things were scalable. For example: the model provides methods that would return all the categories, or all the items for a specific category ensuring a modular design. Even if more categories and items were added in the future, the Model methods would not change and the web application overall would require minimal changes.

The model was also designed with security in mind as well as strict object-oriented programming style and design by contract methodology for the functions it provides.

---

## DEVELOPMENT METHODOLOGY

---

We used an agile development methodology with frequent meetings (scrums and work meetings). Our team roles also shifted around as we all individually found our niche. Even though we had component architects it was still possible to make changes and contributions to other components. As requirements were clarified we were able to adjust accordingly.



---

## SOURCE CODE MANAGEMENT

---

For source code management we used git along with a private repository on bitbucket.org. It allowed us to submit, apply and merge code with minimal effort. Initially it was very challenging to solve the conflicts in the source code but by the end of this project it we are now quite comfortable in solving source code conflicts. Something that we would like to do differently in our future project is to learn and apply concept of branches to our source code management skills.

---

## VIEW DESIGN

---

One of the big things we learned in this project was that we should try to design our view with our own CSS. Using existing templates that forced us to learn challenging programming concepts like AJAX and JS, in which we could not make a lot of progress and took a lot of time. Despite the fancy effects initially it was not worth it as View changes required careful eye to detail and you were guaranteed to break some view feature if you made changes. As a result, we had to scrap the whole idea and start the View from scratch again (twice). We all learned that CSS is very powerful but very time consuming itself as it requires a lot of fiddling around.

---

## SOFTWARE ENGINEERING

---

We learnt that it is very hard to evaluate and estimate proper time management for a feature or a defect for a multi layered/component application. We also learnt that TDD (test-driven design) and testing individual components and feature is very important it allowed us to solve many engineering issues such as calculating extended price in cart and avoided defect at application level which was very important as different components (especially view) could be lagging behind so being able to test a component individually was an invaluable resource.

---

## B2B COMPONENT

---

B2B is the part of project C that run asynchronously with the web server, it consolidates all purchase order from the customer or clients and generate a procurement order.

---

## B2B DESIGN

---

In designing B2b, there were two software that was employed namely SOAP and JAXB.

SOAP was used to connect to the web services of wholesalers who may carry the product that needed to be ordered and return their prices, and return the wholesaler with the cheapest price.

In designing wholesaler, a method was created to find the wholesalers and compare their prices and return the cheapest prices among those wholesaler the carried the product.

We tried to take advantage of existing XML technologies as much as possible (including the XSD and XSL files). JAXB was used to unmarshal the xml purchase orders and marshal the resulting ProcurementOrder object.

---

## IMPLEMENTATION

---

---

### CONTROLLER IMPLEMENTATION

---

---

#### FRONT CONTROLLER

---

In the interest of time we were unable to implement the concept of multiple controllers fronted by a front controller. The front controller would have ensured initialization was done properly and would be the “main” controller that binds to all required URL patterns and forwards to other controllers based on their name (using the Name Dispatcher from the Servlet Context rather than the URL dispatcher from the request).

---

#### SERVER MEMORY

---

To save on memory use, we used minimal data structures – primitive types wherever possible. For example we must keep track of the next order id to use, which is different for each client. So we used a HashMap for this data structure mapping the primitive int clientID (username) to a primitive int order Id. We decided that it was unlikely that we would have more than  $2^{31} - 1$  (max value for int) clients and that any single client would make more than  $2^{31} - 1$  orders. If the website became that popular, we would have to consider even bigger issues such as storage and processor power – we would probably require clusters and advanced system hardware at that point. We consider this outside of the scope of this project. In this case we would use data types like unsigned Integer, Long or BigInteger.

---

#### SEPARATION OF TEXT USED IN VIEW

---

Many Strings used by the View (in JSP files) are poked into the request scope by the Controller. This ensures modularity making it much easier to change Strings in the view as they only need to be changed in one place. Currently the Strings are stored in the Start servlet, but ideally they would be read from an XML file (following a certain grammar) making future translation of the website easier as well and not requiring a recompilation of the core web application when some text needs to change in the view.

---

#### MULTI-THREADING CONSIDERATIONS

---

Since web applications are deployed in multi-thread environments, it is important to ensure our application is thread-safe. In the session scope, we only store things such as the shopping cart and login in the session scope. There would only be issues if each session had multiple controller threads associated with them, which is unlikely but needs to be considered before public deployment.

Another potential problem could be since our clients are companies, multiple users may log in with the same account. This may potentially cause issues not in the session scope but in the context scope. The context scope records the current order id per customer (in a HashMap) and if access to the HashMap is not synchronized across different threads there could be unexpected results (overwritten purchase orders, lost purchase orders, etc). Note that the problem is easier bigger than this, as even different clients logged in will need to update the same HashMap in the context scope through their respective controller threads. To resolve this, a potential

solution could be to use a HashTable instead of a HashMap, as a HashTable is thread-safe (synchronized) whereas a HashMap is not.

---

## VIEW IMPLEMENTATION ISSUES AND SEPARATION OF CONCERNS

---

In the initial design of our website we implemented sliding pages which were functioning in a mobile fashion using CSS3. However, to maintain this sliding page effect at the same time as sending requests to the controller, it was necessary to use JavaScript functions and in particular Ajax technology. As we progressed, we realized that more JavaScript related technologies such as JSON were required to deal with data objects consisting of attribute–value pairs. Integrating various JavaScript methods and technologies took a considerable amount of time and resources and would contradict our timetable to complete the main functionality.

One possible solution was to use JavaScript functions within the jsp page and integrating it with EL statements. Although this method proved to be working, we realized that it violates the separation of concerns principles. As a result, we made a considerable shift from asynchronous web application development techniques to a static method in order to achieve the main targets in our project. However, the infrastructure and templates for implementing asynchronous web application has been considered in the current design of the website. In future modification of the website, pages will be placed in panels (identified by <section> tags using “*st-panel*” class) and transitions will occur among panels through CSS3 features.

---

## CSS3 IMPLEMENTATION

---

Four major categories were considered in establishing the CSS design for the entire website namely:

- **Main:** handling the entire layout and most common elements in the website
- **Header:** controlling all elements in the header such as search box, login form and menu items
- **Categories:** implementing the design and sliding features of the four categories in the main page
- **Items:** containing all (or selected) items for each category or in the view cart

Careful measures were taken in writing our CSS methods to ensure compatibility with different devices and screen resolutions. The key features to achieve this goal were using *percentage* instead of other measuring units (i.e. pixels) and also setting the display attribute to *relative* instead of absolute where necessary.

One of the major issues we encountered during CSS implementation was having identical elements behaviour. Even though these elements were distinguished by classes (and unique IDs in some cases) their intended CSS methods were overlapped by other elements in the upper levels. The way we were able to overcome this problem was to redefine overlapping features so that each element behaves as intended, we also greatly simplified the original CSS we used in order to make it easier to change things in the future.

Another design feature we intended to implement was using media queries to support and provide custom CSS for mobile devices with small screen sizes.

---

## MODEL IMPLEMENTATION

---

---

## B2C SECURITY

---

B2C model will convert the parameter based by controller to required data type for database and also it will perform a validation on the parameter field to prevent SQL injection on the database. In our initial design we were storing password in our client bean for easy transportation in layer but later on we figured out that it poses a big security risk hence we ended up removing the password element from bean and we directly match password to the database without keeping any other copy. In a future implementation we would also ensure that the password is salted and hashed to reduce client impact even if a compromise were to occur. We would also encrypt the user input on the client side so that it is not sent in plain text.

---

## SHOPPING CART IMPLEMENTATION

---

As soon as a client makes a fresh visit to the web application, an instance of cart is created by B2C model to be used until the end of a session. Every single time an item is added by the client, cart will get receive the item number form the model and specified quantity. Cart will itself query the database and get the itemBean and add it the list. By using itemBean it allowed us to store all the values required for displaying the cart. The biggest challenge we faced with cart was designing the itemBean such that it has all the information we need for view delegation as well as shopping cart. For example quantity of particular product and its extended price were stored itself in itemBean. This approach of using an itemBean and ArrayList allowed view to display the cart without any computation including extended price, tax, shipping and total on the fly. All the functionality of cart was tested manually.

---

## PURCHASE ORDER MARSHALLING

---

Once the client confirms the shopping cart and after successful authentication cart once again will be dynamically generated with its content and a purchase order will be generated as an XML file. One issue we found came up with the JAXB marshaller, which would say there were duplicate attribute names in the same class to marshal. This was fixed by explicitly specifying the @XmlAccessorType. Note that the accessor type was not required for Project B which seemed odd. The default Accessor Type seems to be dependent on something outside of our knowledge. We also created an XSL file which will allow customer to view the purchase order in future. One of the most time consuming task in this project was solving repeated values bug in XSL such as “total, shipping, etc”. This issue was solved by creating functions that returned nothing in XSL.

---

## ADVERTISEMENT FILTER

---

Once a specific item is added to task an ad-hoc filter will query the database and poke the advertisement product to the request scope. This product will be displayed to the client in cart along with the option to add to cart.

---

## B2B CONSIDERATIONS

---

A few issues cropped up during the B2B subsystem development. Firstly, how would we find the Purchase Orders directory? Our solution to this was using the class loader of the B2B Main program and using that location to find the Purchase Orders one (it was relative to it). So our B2B program does not require any parameters, all you need to do is run it as a Java application and it does the full procurement for you.

Two additional issues also seemed to be of importance in the B2B part. The first one was to ensure that the same purchase order is not processed by the B2B component twice (procured). In order to do this, we cannot move processed purchase orders folder because of the requirement that it must be accessible to the client at the same URL given when it was purchased. We also cannot change the XML purchase order files too much as it must adhere to the given schema. A solution to this problem could be to get the date submitted from each purchase order and only process it if it was from the previous day (or week) relative to the current date. This date could be recorded in the procurement orders themselves.

The second issue was synchronization issues with the B2C model. It is entirely possible that both components operate at the same time. If this happens, then unexpected results might appear for your Procurement Order. To resolve this, we can use the synchronize method around critical sections.

Note that the way we have implemented B2B makes it very simple to run. After having "model.b2b.runB2B" on the class path, all you do is "java runB2B" and it'll create a procurement report.

---

## THE TEAM

---

---

### DIVISION OF WORK

---

As we had five group members, we broke the project up into components and identified team roles (see Appendix A) which each member ranked from most preferred to least preferred. Then we were all assigned team roles based on what we wanted to do.

**Hosseini:** Component Architect for View component, User Interface Specialist, User Experience Designer, Compatibility tester and Web Developer for the View

**Heba:** Component Architect for the Database component, User Interface Specialist, User Experience Designer, Information Developer Lead, Testing Lead

**Maynil:** Component Architect for the B2C Model component, Component Architect for Filter, Build developer lead

**Osamede:** Component Architect for the B2B Model component, User Experience Designer

**Herman:** Product Team Lead, Product Architect, Component Architect for the Controller component, Component Architect for Analytics

---

### MEETINGS

---

We were all in constant communication through our Whatsapp group conversation, text messages, phone calls, emails, booked study group rooms for meetings, remote study group rooms using IBM smartcloud which allowed video recordings (which recorded our screen sharing and voice) available for those who couldn't make it

How often we met varied widely, estimated to be between 1-5 times a week, between 1 and 13 hours each meeting. Average meeting time I would estimate at 5 hours, average of 3 meetings per week

Consistency maintained by staying in constant communication, not letting anyone get lazy that way.

Lessons learned: Keep it simple initially, do not do fancy view effects until we have a working product meeting all core requirements.

---

## MEMBER CONTRIBUTION

---

Hossein: Designed and implemented majority of CSS (including CSS3 effects), Main JSP pages, implemented JavaScript (later scrapped), Ajax (also scrapped), JSP fragmented view

Heba: Designed and implemented CSS, some JSP pages, the DAO component, search feature, sort by feature, found ¼ of the item pictures, drafted preliminary report, tested and reported bugs

Maynil: Designed and implemented the B2C classes, filter, coordinated source control using bitbucket and helped all members learn git

Osamede: Designed and implemented the B2B classes, found ¾ of all item pictures, provided helper validation methods for B2C model

Herman: Ensured integration between components, performed code reviews, designed and implemented the controller classes, organized team meetings, helped component leads and coordinated project

---

## HOW EACH MEMBER LEARNED THE OTHER COMPONENTS

---

To ensure every member learned all the other components, special regular (weekly) team meetings were set aside for the express purpose of component leads explaining their design and implementation to the other group members

---

## OUTPUT SAMPLES

---

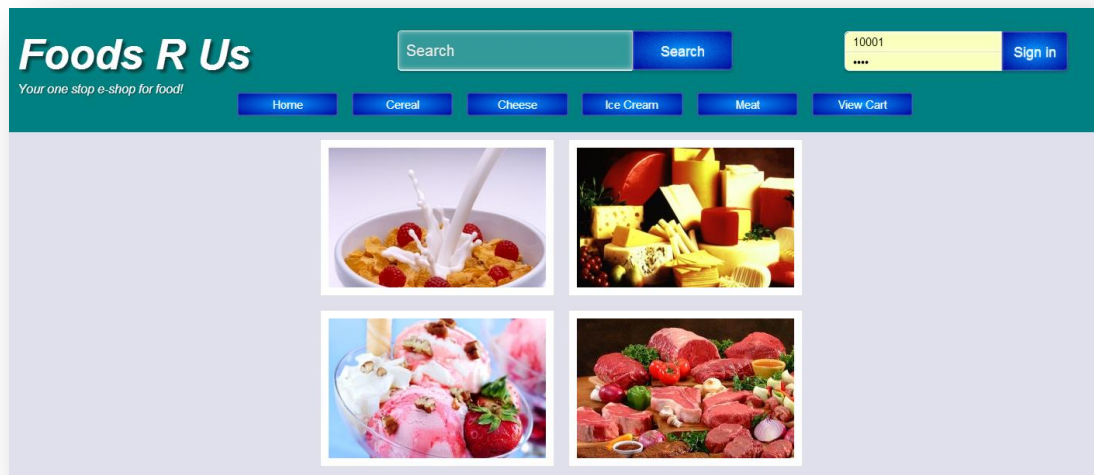


Figure 2: Home Screen

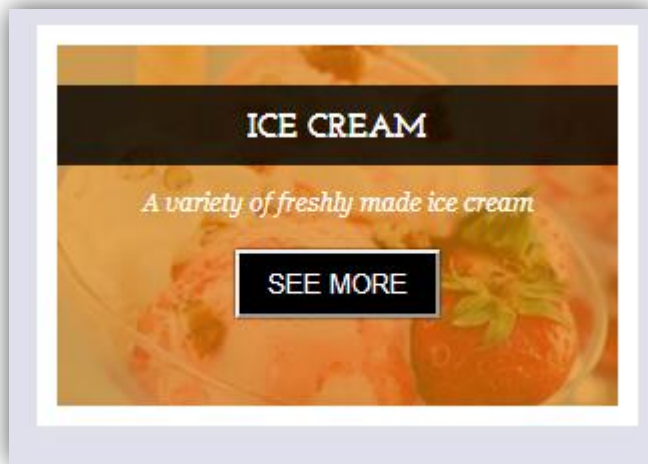


Figure 3: Category on Hover

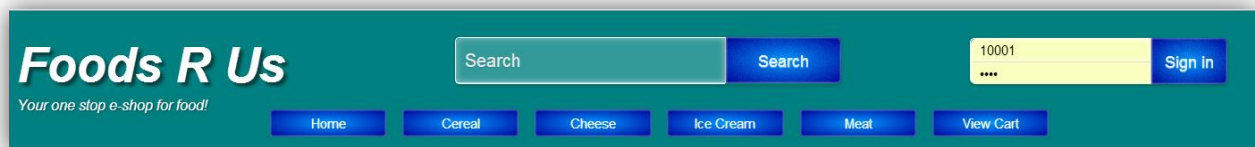


Figure 4: Header for each page

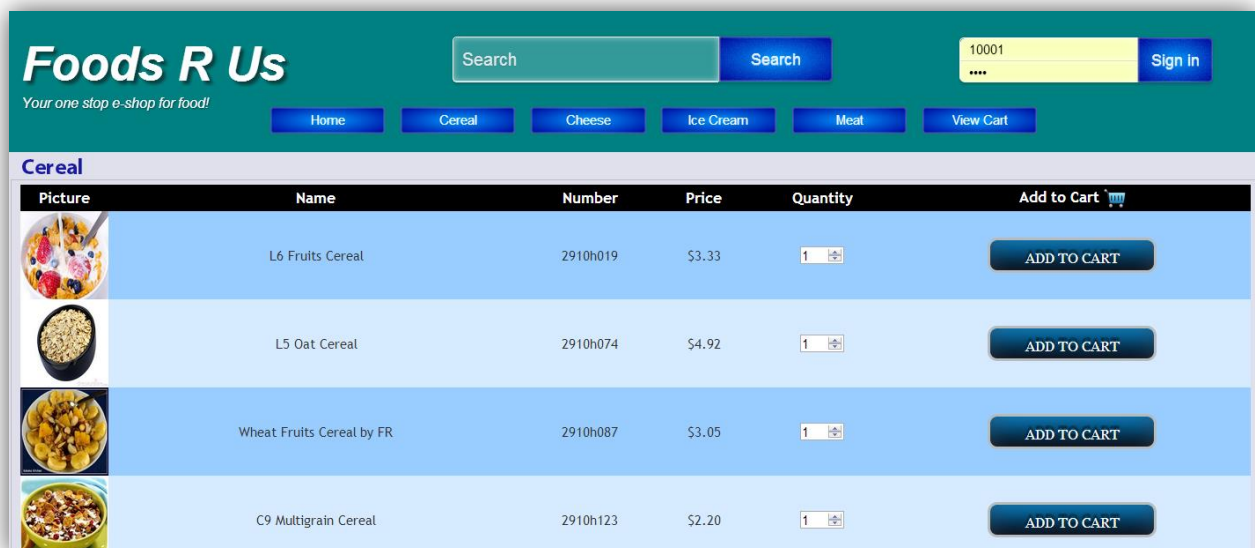


Figure 5: Items page




Picture	Name	Number	Price	Quantity	Add to Cart 
	L6 Fruits Cereal	2910h019	\$3.33	1 	<button>ADD TO CART</button>

Figure 6: Item on hover changes color

Shopping Cart

Name	Number	Price	Quantity	Extended Price
L6 Fruits Cereal	2910h019	\$3.33	1	\$3.33

- Extended total: \$3.33
- Shipping cost: \$5.00
- Tax: \$0.43

Grand Total: \$8.76

CONTINUE SHOPPING

UPDATE CART

CHECKOUT

Figure 7: Shopping Cart

Shopping Cart			
Please confirm your order below:			
Name	Price	Quantity	Total
L6 Fruits Cereal	\$3.33	1	\$3.33
<ul style="list-style-type: none"><li>Extended total: \$3.33</li><li>Shipping cost: \$5.00</li><li>Tax: \$0.43</li></ul>			
Grand Total: \$8.76			
<button>VIEW CART</button> <button>SUBMIT ORDER</button>			

Figure 8: Shopping Cart Conformation page

Order completed
Click <a href="#">here</a> to see your order.
Please save this URL for future reference: <a href="http://localhost:4413/ProjectC/PurchaseOrders/po10001_01.xml">http://localhost:4413/ProjectC/PurchaseOrders/po10001_01.xml</a> . It will also be e-mailed to you for your records.
<button>GO HOME</button>

Figure 9: Successful Purchase



## Purchase Order

**Date: 2013-12-05**

Order Number: 1

**GE Company**

Account Number: 10001

	Item Id	Item Name	Item Price	Item Quantity	Item Extended Price
1	2910h019	L6 Fruits Cereal	\$3.33	1	\$3.33

**Total: \$3.33**

Shipping: \$5.00

HST: \$.43

**Grand Total: \$8.76**

*Figure 10: Purchase Order Receipt*

## Procurement Report

**Date: 2013-12-04**

**Procurement Identifier: 17**

	Item Id	Item Name	Quantity ordered	Winning Price	Chosen Wholesaler	Confirmation message
1	1409S123	E8 Nuts Ice Cream	1	\$4.86	Halifax	Order accepted. Confirmation# T144a74e0
2	1409S811	Brownie Ice Cream with Twirl by BJ	1	\$2.38	Halifax	Order accepted. Confirmation# T144a74e1
3	2002H063	Semi-Cheddar Cheese by JC	4	\$3.43	Vancouver	Order accepted. Confirmation# T144a7421
4	1409S413	K5 Praline Ice Cream	3	\$4.85	Halifax	Order accepted. Confirmation# T144a74e2
5	2002H712	Semi-Monterey Cheese by GK	1	\$3.85	Toronto	Order accepted. Confirmation# T144a749c
6	0905A708	J0 Chicken Meat	100	\$3.75	Halifax	Order accepted. Confirmation# T144a74e3
7	1409S381	Brownie Ice Cream with Praline by MG	120	\$6.06	Toronto	Order accepted. Confirmation# T144a749d
8	1409S004	Fudge Ice Cream with Strawberry by AV	1	\$5.39	Vancouver	Order accepted. Confirmation# T144a7422

*Figure 11: B2B Procurement Report*

## THE SOURCE CODE

Please see the PDF file “CSE4413\_ProjectC\_ScriptKiddies\_SourceCode.pdf” for Source code.

## APPENDIX 1: TEST CASES FOR BEHAVIOR-DRIVEN DEVELOPMENT (USE CASES)

---

---

### TEST CASE #2: A CLIENT ADDS AN ITEM TO THE SHOPPING CART

---

1. The Cart servlet must react by displaying the content of the shopping cart. PASS
2. The display should be tabular with one item per row. PASS
3. The table columns display the number, name, and unit price of each item in a read-only fashion. FAILED[no unit]
4. writable column for the desired quantity and a read-only one for the extended price PASS
5. The display should also indicate a total, a shipping cost (\$5 that is waived for orders of \$100 or more before taxes), HST (13% of total, including shipping if any), and a grand total. PASS
6. The page has three buttons: Update (to refresh the calculated fields after editing a quantity), Continue Shopping, and Checkout. PASS
7. if the entered quantity of an item is zero then it should be removed from the cart. PASS
8. You need to also handle the case of negative or non-numeric quantities. PASS

---

### TEST CASE #3: A CLIENT LOGS IN

---

1. Upon checkout or at anytime, a login form (normally transmitted over https) is served prompting for the client's account number and password. PASS
2. Logout successfully. PASS

---

### TEST CASE #4: A CLIENT CHECKS OUT

---

1. Upon checkout, the controller must ensure the client is logged in PASS
2. must then display a confirmation screen followed by an acknowledgement that the the order has been accepted and is being processed. PASS
3. A URL that the client can visit at any time to view the created P/O must also be provided. PASS

---

### TEST CASE #5: A CLIENT VISITS THE URL OF A P/O

---

1. URL works. PASS

---

## APPENDIX B: INITIAL POSITIONS

---

- Team lead
  - Organize meetings between group members
    - Book study group rooms on campus for face-face meetings and set up audio conference calls and screen sharing for remote meetings
    - Take notes for each scrum and record each remote meeting for those that miss it
    - Ensure that everyone knows what to do and has a say in the team meetings
    - Follow up with each team member to ensure that they know what to do
  - Perform research on the best type of development model to use (Waterfall or Agile). Create a report on findings to be included in the bigger Project C report
- Web Developer
  - Be familiar with 25 most dangerous software errors: <http://cwe.mitre.org/top25/>
  - Two component architects: one for the Model component and one for the Controller component
- Database administrator and specialist
  - Program the DAO, work with the Model component architect when designing your JavaDoc API
- White-box Tester/Functional verification team
  - Be familiar with 25 most dangerous software errors: <http://cwe.mitre.org/top25/>
  - Perhaps using JUnit
- Black-box Tester/Software verification team
  - Be familiar with 25 most dangerous software errors: <http://cwe.mitre.org/top25/>
- Performance tester
- User Interface specialist & component architect
  - Develop the View component. Ensure proper comments and documentation. JavaDoc annotation where appropriate and include author name for all implemented pages
  - Familiarize yourself with View technologies such as CSS, JSP, Jasper,
- User Experience designer
  - Make sure the website is intuitive and fun to use
- Build developer
  - Compare different source/version control mechanisms (git vs svn) and if we should use a service (Github, Bitbucket, etc)
- Security specialist
  - Ensures that the software contains no security holes
    - Make sure we avoid the most dangerous software errors (<http://cwe.mitre.org/top25/>)
- Information developer
  - Finalize report
    - Review the reports from each individual team member and
- Privacy and ethics advocate
  - Ensure our software meets the privacy by design principles here: <http://www.privacybydesign.ca/index.php/about-pbd/7-foundational-principles/>
  - Consider any possible ethical implications of our product
- Translation team and translation component architect

- Create framework for easily translated messages (don't hardcode messages/Strings, get from some sort of file, like XML or properties file)
- Development manager
  - Ensure our goals are realistic and deadlines can be met.
  - Ensures the product has a business purpose.
- Architect
  - Know the technology; should be able to be the final point of contact for technical questions
    - Know how the following frameworks and technology work: Java EE, Javascript, HTML, CSS, XML and related technologies (JSON, properties file), Ajax, PHP
    - Know database architecture and be aware of issues (concurrency, etc)
    - Have knowledge of the evolution of the web from web 0.0 to 3.0, how our project fits into it and in the future how to make it adhere to future web standards such as SOA
    - Ensures the software is designed in an extensible way for ease of future improvements
    - Write reports on all the above
- Chief Architect
  - Technically responsible for multiple products
  - Position held by Professor Hamzeh Roumani
- Legal team
  - Ensure intellectual property theft does not occur
    - Examine the different possible licenses and the ones relevant to the technologies we are using
    - Ensure no copyright violations occur; no code should be copied – all should be original material

#### Development best practices

- <http://cwe.mitre.org/top25/>

---

## APPENDIX C: B2C CLASSES

---

### B2C View Delegation

B2C model reacts to controller's request, which includes initial and fresh visit to the web app, populating a particular category selected by the client.

### B2C Client authentication

B2C model is responsible for taking username and password for client and authenticating its identity against database.

### Cart

Cart is an ArrayList of itemBeans responsible to keep the track of food items added by client, update the quantity of one or more item as well as any removed item. Cart is designed in a way that every time a change is made by the client it will recalibrate it self with all the required fields automatically. This allows us that no computation is required to be done by neither controller nor view.

### Purchase Order Marshaling

B2C model is outfitted with a XML Marshaller to marshal purchase orders to an appropriate XML file for B2B model as well as allowing client to view their order in future.

### Advertisement Filter

Our web app is equipped with an ad-hoc filter that will display advertisement only when a specific product is added to the cart.

---

## APPENDIX D: ORIGINAL PROJECT TIMELINE

---

---

### PROJECT TIMELINE

---

The project timeline will consist of three iterations. Each iteration is one week long will start on Tuesday and end on Monday. Our Milestone deadlines will be on the Mondays at 7pm during our regular face-to-face Monday meeting from 7-10pm. Other than the three iterations, we also have a Prototyping phase before our first iteration and a Final Release phase after our third iteration.

---

### PROTOTYPING PHASE

---

**Dates:** November 10<sup>th</sup> – November 11<sup>th</sup>

**Objectives:** All component architects should create a semi-working prototype/concept (perhaps just a proposed API) for our initial face-face meeting on November 11<sup>th</sup>, 7-10pm.

---

### ITERATION 1

---

**Dates:** November 12<sup>th</sup> – November 18<sup>th</sup>

**Objectives:** Integrate components, divide work

**Milestone 1 exit criteria:** Basic layout and APIs done

---

### ITERATION 2

---

**Dates:** November 19<sup>th</sup> – November 25<sup>th</sup>

**Objectives:** Core B2C functionality done

**Milestone 2 exit criteria:** B2C use cases done

## ITERATION 3

---

**Dates:** November 26<sup>th</sup> – December 2<sup>nd</sup>

**Objectives:** Adhere to Project C requirements

**Milestone 3 exit criteria:** All requirements completed

## FINAL RELEASE PHASE

---

**Dates:** December 2<sup>nd</sup> – December 4<sup>th</sup>

**Objectives:** Finishing touches and productization

## APPENDIX E: DETAILED WORKFLOW

---

Client visits a Foods R US page. Client will see the home page with the four categories centered with a “See More” Button that is displayed on hover that takes the client to the Items.jspx page. The main page also include other JSP include pages such as Header and Footer.

Header includes a search bar, login section, Menu Bar, and Foods R Us logo. The header is identical for all jsp pages.

While using search, client can type anything and search in our webapp returns the list of items where the name of the item matches with the text entered.

Client can navigate to items using menu bar located in the header of each page.

Client can login at the beginning of the session or at any point in time. Login feature/Logout is available across all pages.

Client can visit Item.jspx page by either clicking at the menu bar or by clicking on see more from the main category centered menu.

In Items.jspx, client can see a list of items based on the category selected. Client can add quantity, reduce quantity and add to cart by using “add to cart” button the redirect the client to the cart view.

In Cart view, we have enabled three buttons where client can “continue Shopping” [client will be navigated to home page], “Update Cart” [Client can add more quantity of the items that are selected for purchase].

And “Checkout” [Client will be prompted to login if they didn’t do so, else go to confirmation View]

Once the client confirms the purchase order, client is automatically logged out of the session and must login again if they wish to purchase extra items.

## APPENDIX F: ENHANCEMENT BACKLOG

---

- Only allow logging in 5 times max, then a log in throttle with timer
- Pretty URL to allow clients to access categories/items directly

- Resize images before getting to view to decrease bandwidth costs
- Media queries to support mobile platforms (view)
- Sort by name, price, etc for items
- Pagination of items page
- Translation part where all text is pulled from an XML file. Translate to a second language for proof of concept
- View overhaul to use AJAX and JavaScript, and simplify CSS as much as possible
- Modularizing the controller: front and others
- B2B synchronize purchase orders with B2C
- B2B ensure same purchase order is not procured again.