EECS 4413
November 28th 2014
Project C
Yuiping Lin, Dominique Luna, Thalammeherage Perera

Table of Contents

Design

B2C

Frontend

The design of our app focuses on a very minimal UI. The product itself is to be able to select and order food; we strived this process to be as seamless as possible. We also desired to avoid reloading complete pages for any action, and for this reason implemented as much of a single page app design as we could. For the unfamiliar, a single page app design makes heavy use of AJAX to change components of a page vs. having multiple pages. In practice this allows content to be more reactive, and in the process UX is drastically improved. In fact the only page completely different from the main page is the checkout page.

We argue our design is an extension of the MVC pattern, the difference being the view is far more reactive. The client receives immediate feedback on their actions. In order to realize the idea of a AJAX heavy architecture we made our routes API endpoints. For example *eFoods/backend/category/3* returns all items associated with the category where id = 3. We then form data flow paths between view elements (html) and the corresponding endpoint. This is done through data attribute tags in html elements. We have multiple small jspx files to keep this organized and maintainable.

Another example of this is when the client adds an item to the cart, the cart number is automatically updated, there's no need to refresh the page! The design if reflected further in the search bar. We could have done this a couple different ways:

1) Have items names appear via dropdown during typing
2) Have the main content automatically reload during typing

Due to our AJAX heavy design we felt the second option was better aligned with the overall design. Option 1 has some issues, the main one being it adds clutter and adds very little functionality over option 2. We also argue that option 1 conveys less information than option 2, due to option 2 showing the full item description in the main content section of the page. Lastly as mentioned previously option 2 far better aligned with our original design. It's simply a filter over the AJAX results ,thus presenting results in an identical manner.

Backend

We opted to use Front Controller Pattern because of the sheer number of URL endpoints that we were required to serve and the authentication was a concern. Thus all requests backend code is

routed through our front controller which in turn checks for the validity of the requested route as well the if the request requires the user to be authenticated. As most of the requests to backend were made as AJAX requests, the front controller helps in identifying the type of the request and respond accordingly (i.e: partial HTML vs full HTML).

Since the database provided for the project was strictly read-only it was decided to use the file system to store purchase order information. Each PO is saved as an individual XML file in a designated folder. Since indexing the POs on the disk is an expensive task we decided to use easily identifiable file names to facilitate faster searching. The scheme that we use on the app is as follows

"[Purchase Order Id]_[Account Name]_[Purchase Order Status].xml"

- Purchase Order ID: Next highest order ID that is not saved to the disk.
- Account Name: Name of the user that created the order
- Status of the Order: One of 'new', 'pending', and 'purchased'

Apart from outputting HTML, the app outputs XML and JSON data as well. XML output is generated when the B2B requests all available orders and JSON is produced to notify the frontend of any errors in case of AJAX requests.

We also decided to provide the customers with the convenience of tracking the status of their order. This also provides the added benefit that this allows multiple B2B instances to poll for POs from the main application, as they will only attempt to process files marked as 'new' opposed to 'pending' or 'purchased'

B2B

The primary principle we had in mind in designing the architecture of B2B is "Separation of the Concerns". Since the two major tasks of B2B part are consolidating the purchase orders (P/O's) and procuring them from the wholesalers, we decided to make two classes Aggregator and Purchaser to perform these tasks correspondingly. The web service can be viewed as a subtask of procurement and it could be performed within Purchaser. However, we think that it would be better to provide a constant wholesaler interface for Purchaser. In this case, our application can accommodate wholesalers with different WSDL and Purchaser do not need to remember them. Hence to take these advantages and to further concentrate Purchaser in its own task, we wrap the details of WSDL into Wholesaler and provide it as an ordinary portal to Purchaser. This also gives us an extra benefit of dynamically adding, changing and removing wholesalers from the wholesaler name list in Purchaser. Similar to above, we also decided to wrap the P/O files downloading details into PODownloader to further concentrate Aggregator's concern.

As for the connection between B2B and B2C, because the communication between B2B and B2C are limited, we need to have a stable portal for B2B to access files in B2C. However, we also want a more flexible way to store P/O files and identify their status. To achieve this goal, we finally came up with the idea of downloading P/O files in two steps. In the first step we get the information of P/O files from a stable portal. Then by using the information from the first step, we can access the files in B2C more flexibly.

AUTH

When designing the Authentication module we primarily considered security, ease of use and reliability. When the user clicks on the "Log In" button from the main application, we ensure that user is always redirected to the "HTTPS" URL of the authentication page. Upon landing on the authentication page, the user presented with a clean HTML user interface that clearly shows where they would enter their credentials. As it is a decoupled system, we have made sure that the user is distinctly informed what authentication system they are getting verified against.

After user submits the form, the system will attempt to authenticate the user. On failure it will show a banner in color red stating the reason for failure and prompting them to try again. Upon success, the system will redirect the user to a preconfigured endpoint in the main application with the information about the user. The architecture of the Authentication module is designed in a way that we account for errors in the authentication module itself and remote authentication services such that the user may always be informed about the status of their request.

Lassonde Auth

@ Username

@ Password

You will be authenticated against EECS LDAP.

Login

User Interface for AUTH

Implementation

B2C

Frontend

The implementation of our MVC reactive architecture starts with the front-end AJAX component. We rely heavily on data attributes in html <form> and <a> elements to deliver information about which sections of the view need to be updated following the action of sending the form(<form>) or clicking the link(<a>).

Ex:

```
<h1>
        <a href="backend/category/${category.id}" data-refresh-ids="main-content" onClick="return eFoods.app.handleHref(this,
true);">${category.name }</a>
</h1>
```

Here we can see the *data-refresh-ids* attribute. This is where we specify which parts of the view need to be possibly changed after the action takes place. We can also see the *onClick* attribute, here we're calling the *handleHref* function, the other main function of this kind is *handleForm*. The third component to this <a> element is the *href* attribute. This represents the API endpoint we're going to receive the data form, in this case parameterized by ${category.id}.

When this link is clicked what happens is the *handleHref* function is called on the <a> tag itself. We then take the ids from the *data-refresh-ids* attribute and call the endpoint specified in the *href* tag via AJAX. On return, the main-content section of the view is then refreshed with the data received from the endpoint. The flow is very similar for a form except we also package up the form input elements along with attributes.

Ex:

```
<form method="POST" action="backend/cart" data-refresh-ids="cart-info, item-${item.number }"
        onsubmit="return eFoods.app.handleForm(this);" ……. (this section is excluded for brevity)
```

In this example we can see *data-refresh-ids* has two ids: "cart-info" and "item-${item.number}". This form is used whenever an item is added to the cart. Through the ids we can identify two sections of the view will change. When we add an item, the section "cart-info", which shows the cart icon and a number of items will change, as well as

the item containing the form itself. Here the *action* attribute signifies the API endpoint and we request using POST.

The search bar uses much of the functionality described above. There is however an additional detail that we really had to get correct in the implementation. The naive approach is to have an event listener on the input which sends an ajax request as described above on "keyup". The problem with this is a request is fired each time the client hits any key.

Ex.
    The query "sirloin meat" will use 12 requests, one for every character.

This is a huge waste of resources, especially when the final query is perhaps the only worthwhile one. Our solution to this is to use browser timeouts to send the requests. Concretely we send a request on each timeout. Notice, this in itself still isn't a great solution. If the timeout window is too small we're still sending many requests, if it's too large it will feel slow.

To solve this issue we use the *clearTimeout* function. Whenever this function is called on a timeout it resets the timeout. For example if timeout A had 50 ms left before it fired and it starts from 400 ms, *clearTimeout* will set it back to 400 ms. Thus for final implementation is to reset the timeout each time the client hits a key. We set the timeout to 400 ms. This is long enough to not override the typing speed of the client, but short enough to have a responsive feel.

<u>Pitfalls</u>

1. Due to the items being programmatically generated after ajax requests it's more difficult to query the elements through the DOM. An example of this is adding event listeners, it's still possible but requires more effort.

Backend

The entire application code is mounted at the url "/backend/*" which forwards all requests to the front controller. This is the only URL that the Tomcat is configured to use.
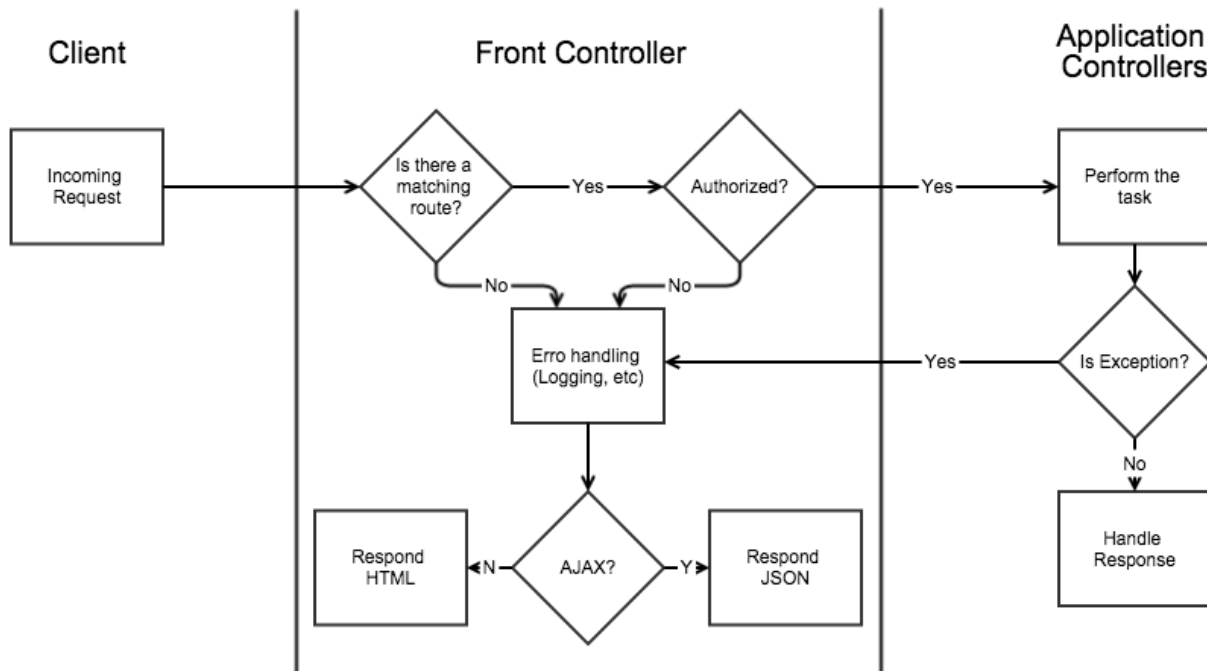
Application Request Lifecycle

Request Router

It was quite apparent that the app would require a lot of URL endpoints, some requiring authentication. As using 'if' conditions to check for each route seemed tedious we decided to implement a custom request router.

The request router would hold a set of routes that containing a URL pattern – a regular expression that corresponds to an incoming request, a HTTP method – one of "GET" or "POST", the destination controller, the action – identifying the sub task of the controller and whether the request requires the user to be authenticated.

We configure these "routes" at the initialization of the app and poke the router in the context scope. For every incoming request, the router will match and forward the request to the corresponding URL pattern and HTTP method. Further more via the use of regular expression groups, we were able to extract parameters out of the URL with ease. This helped a lot in streamlining the development process.

```
appRouter.addRoute(new Route(
    "^/item/(?<itemNumber>[0-9a-zA-Z]+?$",  //Regex for URL
    "Item",                                 //Destination Controller Name
    Route.METHOD_GET,                       //Incoming request type
    Item.ROUTE_BY_NUMBER_FULL,              //Action of the controller
    False                                   //Requres authentication
));
```

Code used to add a request router

<u>Handling Ajax requests</u>

The frontend's javascript is configured to add an extra GET variable "ajax=1" to every request it makes via AJAX. This helps the backend to identify the nature of the response. This was especially useful when handling exceptions, as the backend should know to output in the correct format (JSON vs HTML) depending on the request type.

<u>Single Sign On</u>

As the application requires the user to be redirected to a different server for authentication, a method to keep track of state was needed. This was a challenge as a single page app, it would never change its URL. Thus we decided to keep track of the current state of the app (the information user is currently look at) by changing the document hash every time. In this way, once the user is signed in and redirected back to the site, the app would know exactly where to redirect the user to thus resuming the flow. This gave us the added benefit that the application was now bookmark-able by clients.

"http://efoods.herokuapp.com/#!backend/category/4"
Sample Application URL with the bookmark-able document hash

<u>Purchase Order Files</u>

The main problem with handling Purchase orders was that we used the file system as the storage medium. Not only it is slow, it was difficult to index. As a solution to address this issue we enforced strict file naming and path conventions, and re-indexed the filenames on every request to ensure we don't overwrite anything.

Limitations

Purchase Order System Performance
We use Java's synchronized classes to perform thread safe operations on the file system. This is a massive performance overhead and we have experienced it to be unreliable on certain file systems and operating systems.

B2B Authentication
At it's current state the B2B URL endpoints provide no authentication. Although it was not specified in the requirement document explicitly we believe that it is good practice to implement this functionality. A one way to provide this feature would be through a shared token in every request.

Admin Users
App has the fuctionality to detect admin users but for the submission we have forced all uers to be admin for the sake of testing. (As we do not know the usernames of the project markers)

Pagination
Application does support pagination of Item listings when supplied with GET parameters "limit=?" and "page=?". However due to time constraints, we did not implement this feature on the frontend.

Application variables
We make use of many static variables to change the functionality of the app. However due to time constraints we were not able to externalize these strings to the context.xml file to provide customizability to the client.

B2B

Thanks to the clear design of the architecture of B2B, there was no much difficulties during the implementation except for unmarshaling. At first, due to misunderstanding of the given schema, we only use three bean classes POBean, CustomerBean and ItemBean for unmarshaling. However, it does not give out the expected result and the items are not unmarshaled as a list of ItemBean's in POBean. By carefully investigating the schema and comparing the generated result and the given example, we finally realized that "item" is an element sequence of "items" and "items" itself is an element of "order". Hence we made an extra bean class ItemsWrapper that contains the list of ItemBean's and problem solved. By solving this problem, we got an better understanding of correspondence between xml element and bean class, that is, each xml complex type in the schema corresponds to a new bean class.

We claim that our B2B part fulfills all the project requirements and is capable of automatically identifying new P/O's in B2C, downloading them, aggregating and procuring them, and finally generating the procurement report, as a continuous process. This continuous process is realized by the following steps:

1. A portal in B2C is provided for B2B to access the information of new P/O files (encoded as an .xml file);
2. B2B downloads the information file;
3. B2B downloads the new P/O files using the URLs in the information file;
4. Aggregator takes these P/O files and generates an aggregated order;
5. Purchaser takes the aggregated order, do the procurement and finally generates the procurement report.

AUTH

The authentication module consists of 3 primary components.

1. Business logic and View Renderer
   - · This is a PHP script that inspects incoming request and validates for its authenticity. The validation is performed via a signature verification is discussed below. Upon form submission, it will read the application configuration and perform the authentication and redirection accordingly.
2. Configuration
   - · This is a protected JSON object that contains the parameters necessary for the business logic to function. The configuration may include shared authentication keys, paths and URLs for redirection, etc.
3. Security challenge
   - · A text file containing a random string that is placed in a folder secured by an Apache htaccess file. Authenticator will attempt to read this via a HTTP request using the provided credentials. On success, it will use the file system to read the actual content of the file and verify that HTTP call was authentic. This is used thwart DNS based attacks or issues related to false configuration.

Authentication Flow

A key is shared between the main application and the authentication module. When user clicks the "Log In" button from main application, the application will redirect the user to the Authentication module with a payload signed with the shared key. Authentication module is then able to verify that the request was originated from the eFoods application and then present the user with the login page.

As HTTP Basic Auth against Apache is the only authentication option provided to us by the requirements, upon form submission Authentication module will attempt to access the security challenge folder through HTTP. We use php-curl extension that is built-in to PHP to perform this. If the request returns non-success response the user is informed and prompted to try again. If successful, the contents of the response are the compared against the actual content of the security challenge file. Once the checks are passed, we are confident that the user is who they claim to be. Thus a signed payload will be generated consisting user's full name and the user name they used to sign in. We use the UNIX command 'getent' to query users full name from the name service. The payload will be forwarded to a pre-specified endpoint (by the configuration file) in the main application where it will be verified for authenticity and then accepted.

**"nonce=a34d32bdaf&username=cse11011&fullname=Rajitha+Perea"**

A payload before being encoded.

http://efoods.herokuapp.com/backend/login/authenticate?payload=dXNlcm5hbWU9Y3NlMTEwMTEmZnVsbG5hbWU9Um
FqaXRoYStQZXJlcmEmbm9uY2U9OUJFMDU0N0MxMDI3RkRDRDZGMUY1NjEwMzIwMjlENDU%3D&signature=0096e49cb6f
12ed9f163dc726966226e6fa4073a8e4034d4730165

The actual redirect back with signed and encoded payload as GET parameters.

Security Considerations

One of the main flaws of the initial design was that although it guarantees the integrity of the request; it does not guarantee the authenticity. A person could capture the redirect request and replay it to trick the application in to logging them in. To work around this issue a nonce was generated with every login request and sent to the authentication module. This nonce – a random alphanumeric string will stay in the session scope of the main app for a limited time (10 minutes) and then expire. The Authentication app will append this nonce value to the redirect payload and thus the main application is able to verify it against the nonce in it's session. Upon nonce verification on main application side the nonce is immediately expired thus guarantees that the request is not replayable.
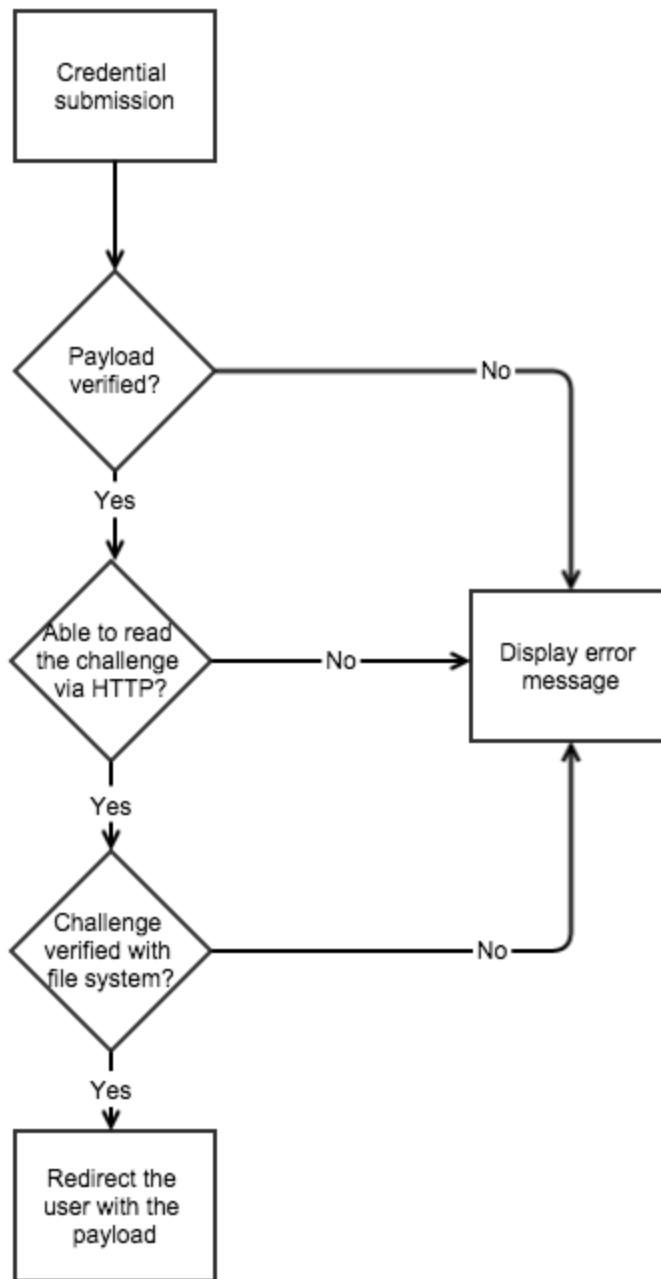
Authentication flow

The Team

Collaboration

The work of this project are divided amongst team member as follow:
Yuping Lin : the B2B part and the analytics feature of B2C part;
Dominique Luna: the frontend and collaborates in the implementation of backend;
Thalammeherage Perera: the AUTH part and the main portion of backend

We had an initial meeting where we discussed the design of the app, what we wanted to accomplish, the responsible part of each member etc. After the initial meeting we didn't have further full in person meetings. But we formed a Google Hangouts chat that we **heavily** used as well as a private Github repository for the project, this is where we collaborated on code.

In this project, we learned a lot about making a working real world project by combining the knowledge and techniques we learned from course and as well the technique not taught in class like AJAX. Further more, we also learn to how to collaborate with team members and maintain consistent progress. We think Github we used in this project is a very powerful tool to maintain consistency in team work.

Yuiping Lin

I am mainly responsible for the design and implementation of the B2B part. I designed and implemented the whole B2B part, with some help from other team members. And I also implemented the analytics features in the B2C part.

I learned and understood the other parts of the project like B2C part and the frontend by reading through the codes we shared in Github, and by discussion with the other team members. Through the discussion, I understood the other parts of the project about how they work and why they are designed and implemented like that. By far, I should claim that I fully mastered the B2B part, and understands most of the B2C and Auth part. Though I am less familiar with the frontend part like using the AJAX, I will try my best to understand it in the next few days.

Dominique Luna

I mainly focused on the B2C aspect of that app. I worked on the majority of the frontend, html, css, js. I also made contributions to the backend sections of the B2C app, such as implementing search, parts of checkout and xml/xslt for PO orders. Worked on bugs that arose in various parts.

I learned about the other sections of the project by reading through the code on Github and asking fellow team members questions about parts I wasn't familiar. As of the writing I'm comfortable with the B2C portion of the app. The B2B and AUTH portions I'm less familiar with but will try to get familiar with them by the lab test date.

Thalammeherage Perera

I contributed primarily to the design and the architecture of the B2C app. I was also responsible for the core implementation of the backend including the custom routing engine, shopping cart, and purchase order handling. Implementation of the Authentication portion of the web app was done by me as well and it is currently hosted at my EECS web root. Furthermore, I implemented the cross-selling feature and contributed to frontend javascript code to implement the AJAX partial loading functionality.

I learned the portions of the project that was not done by me by looking through git commit logs and discussing with my group mates. I believe the discussions heavily influenced the overall quality of the application as well as the knowledge of the app among the members. I also learned about certain aspects of the app by finding and removing bugs.

Declaration

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

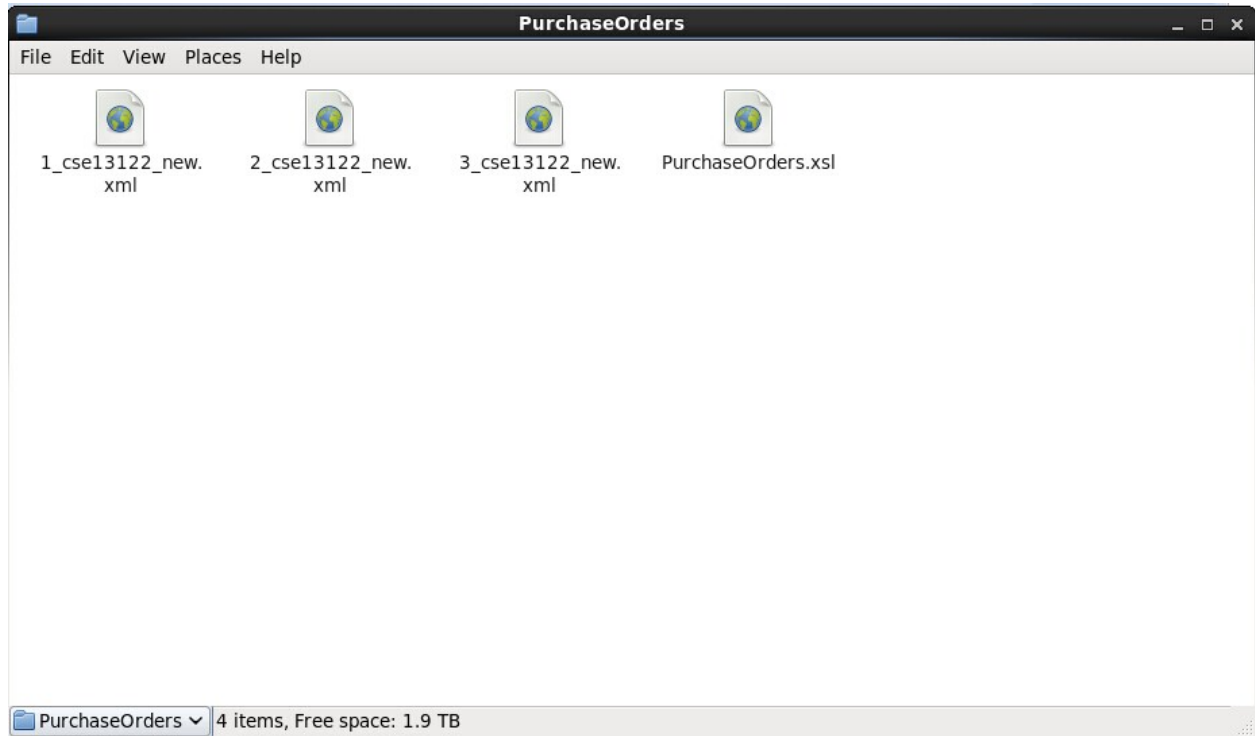*Thalammeherage Raj Perera*
Dominique Luna
Yuiping Lin

Output Samples

We tested the B2B part on three purchase orders. These three orders are show in the server directory of B2C part as:
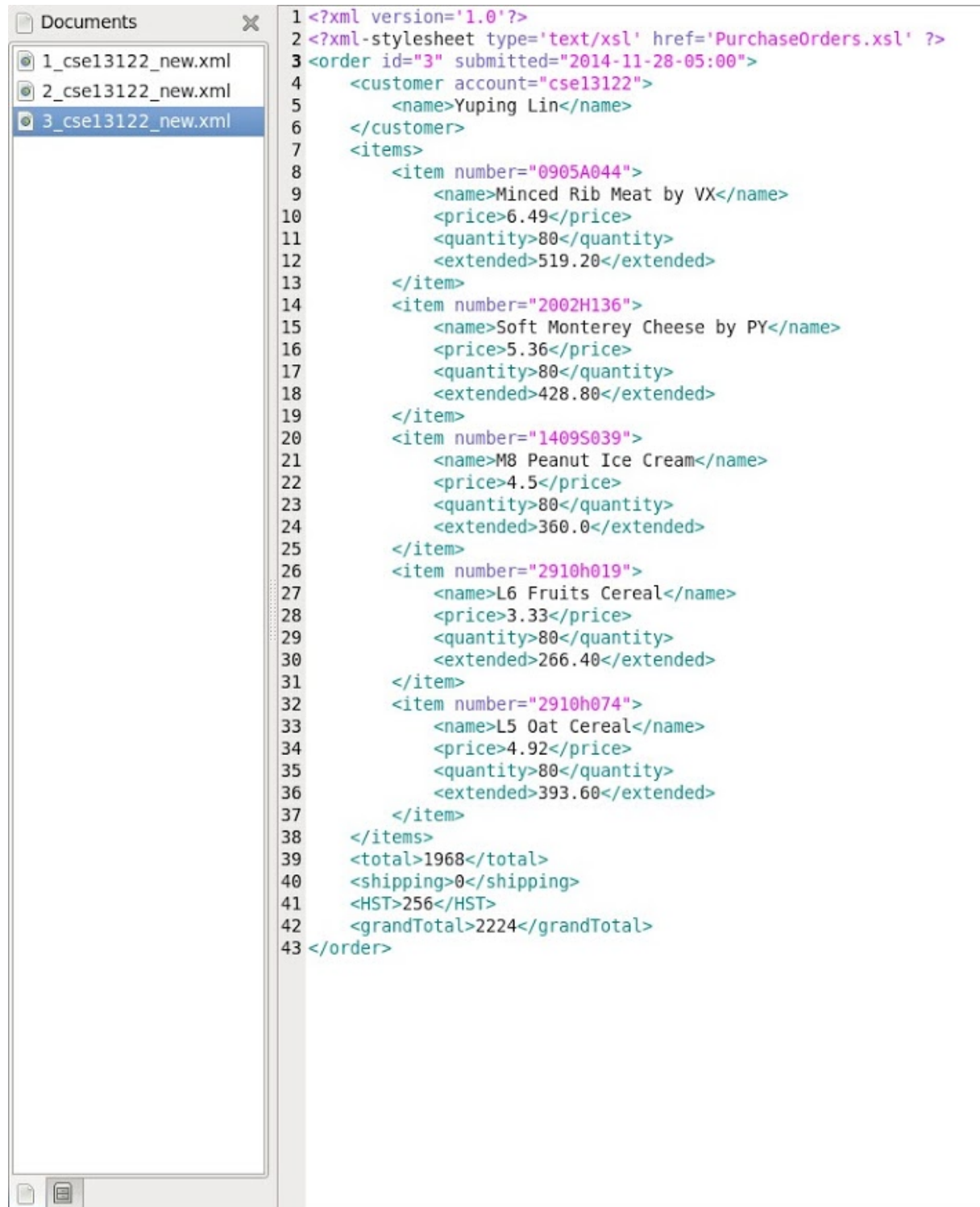
The content of the files are:

1_cse13122_new.xml

```xml
<?xml version='1.0'?>
<?xml-stylesheet type='text/xsl' href='PurchaseOrders.xsl' ?>
<order id="1" submitted="2014-11-28-05:00">
    <customer account="cse13122">
        <name>Yuping Lin</name>
    </customer>
    <items>
        <item number="2002H123">
            <name>Provolone Cheese by RI</name>
            <price>3.84</price>
            <quantity>100</quantity>
            <extended>384.00</extended>
        </item>
        <item number="1409S004">
            <name>Fudge Ice Cream with Strawberry by AV</name>
            <price>7.15</price>
            <quantity>100</quantity>
            <extended>715.00</extended>
        </item>
        <item number="2910h019">
            <name>L6 Fruits Cereal</name>
            <price>3.33</price>
            <quantity>100</quantity>
            <extended>333.00</extended>
        </item>
        <item number="2002H063">
            <name>Semi-Cheddar Cheese by JC</name>
            <price>4.26</price>
            <quantity>100</quantity>
            <extended>426.00</extended>
        </item>
        <item number="0905A044">
            <name>Minced Rib Meat by VX</name>
            <price>6.49</price>
            <quantity>100</quantity>
            <extended>649.00</extended>
        </item>
        <item number="0905A112">
            <name>T6 Sirloin Meat</name>
            <price>6.01</price>
            <quantity>100</quantity>
            <extended>601.00</extended>
        </item>
    </items>
    <total>3108</total>
    <shipping>0</shipping>
    <HST>404</HST>
    <grandTotal>3512</grandTotal>
</order>
```

Documents
- 1_cse13122_new.xml
- 2_cse13122_new.xml
- 3_cse13122_new.xml

2_cse13122_new.xml

Documents ✕

- ⊙ 1_cse13122_new.xml
- ⊙ 2_cse13122_new.xml
- ⊙ 3_cse13122_new.xml

```xml
1  <?xml version='1.0'?>
2  <?xml-stylesheet type='text/xsl' href='PurchaseOrders.xsl' ?>
3  <order id="2" submitted="2014-11-28-05:00">
4      <customer account="cse13122">
5          <name>Yuping Lin</name>
6      </customer>
7      <items>
8          <item number="0905A123">
9              <name>Fine Beef Meat by PQ</name>
10             <price>6.65</price>
11             <quantity>50</quantity>
12             <extended>332.50</extended>
13         </item>
14         <item number="0905A112">
15             <name>T6 Sirloin Meat</name>
16             <price>6.01</price>
17             <quantity>50</quantity>
18             <extended>300.50</extended>
19         </item>
20         <item number="2002H123">
21             <name>Provolone Cheese by RI</name>
22             <price>3.84</price>
23             <quantity>50</quantity>
24             <extended>192.00</extended>
25         </item>
26         <item number="2002H136">
27             <name>Soft Monterey Cheese by PY</name>
28             <price>5.36</price>
29             <quantity>50</quantity>
30             <extended>268.00</extended>
31         </item>
32         <item number="1409S039">
33             <name>M8 Peanut Ice Cream</name>
34             <price>4.5</price>
35             <quantity>50</quantity>
36             <extended>225.0</extended>
37         </item>
38         <item number="2910h019">
39             <name>L6 Fruits Cereal</name>
40             <price>3.33</price>
41             <quantity>50</quantity>
42             <extended>166.50</extended>
43         </item>
44     </items>
45     <total>1485</total>
46     <shipping>0</shipping>
47     <HST>193</HST>
48     <grandTotal>1678</grandTotal>
49 </order>
```

3_cse13122_new.xml

Documents

- 1_cse13122_new.xml
- 2_cse13122_new.xml
- 3_cse13122_new.xml

```xml
1  <?xml version='1.0'?>
2  <?xml-stylesheet type='text/xsl' href='PurchaseOrders.xsl' ?>
3  <order id="3" submitted="2014-11-28-05:00">
4      <customer account="cse13122">
5          <name>Yuping Lin</name>
6      </customer>
7      <items>
8          <item number="0905A044">
9              <name>Minced Rib Meat by VX</name>
10             <price>6.49</price>
11             <quantity>80</quantity>
12             <extended>519.20</extended>
13         </item>
14         <item number="2002H136">
15             <name>Soft Monterey Cheese by PY</name>
16             <price>5.36</price>
17             <quantity>80</quantity>
18             <extended>428.80</extended>
19         </item>
20         <item number="1409S039">
21             <name>M8 Peanut Ice Cream</name>
22             <price>4.5</price>
23             <quantity>80</quantity>
24             <extended>360.0</extended>
25         </item>
26         <item number="2910h019">
27             <name>L6 Fruits Cereal</name>
28             <price>3.33</price>
29             <quantity>80</quantity>
30             <extended>266.40</extended>
31         </item>
32         <item number="2910h074">
33             <name>L5 Oat Cereal</name>
34             <price>4.92</price>
35             <quantity>80</quantity>
36             <extended>393.60</extended>
37         </item>
38     </items>
39     <total>1968</total>
40     <shipping>0</shipping>
41     <HST>256</HST>
42     <grandTotal>2224</grandTotal>
43 </order>
```

These file are shown in the browser as follow: (with xlt style sheet)

1_cse13122_new.xml

Order Information

ID: 1

Time Submitted: 2014-11-28-05:00

Customer Name: Yuping Lin

Item Number	Item Name	Price	Quantity	Extended
2002H123	Provolone Cheese by RI	3.84	100	384.00
1409S004	Fudge Ice Cream with Strawberry by AV	7.15	100	715.00
2910h019	L6 Fruits Cereal	3.33	100	333.00
2002H063	Semi-Cheddar Cheese by JC	4.26	100	426.00
0905A044	Minced Rib Meat by VX	6.49	100	649.00
0905A112	T6 Sirloin Meat	6.01	100	601.00

Costs

Total: 3108

Shipping: 0

HST: 404

Grand Total: 3512

2_cse13122_new.xml

Order Information

ID: 2

Time Submitted: 2014-11-28-05:00

Customer Name: Yuping Lin

Item Number	Item Name	Price	Quantity	Extended
0905A123	Fine Beef Meat by PQ	6.65	50	332.50
0905A112	T6 Sirloin Meat	6.01	50	300.50
2002H123	Provolone Cheese by RI	3.84	50	192.00
2002H136	Soft Monterey Cheese by PY	5.36	50	268.00
1409S039	M8 Peanut Ice Cream	4.5	50	225.0
2910h019	L6 Fruits Cereal	3.33	50	166.50

Costs

Total: 1485

Shipping: 0

HST: 193

Grand Total: 1678

3_cse13122_new.xml

Order Information

ID: 3

Time Submitted: 2014-11-28-05:00

Customer Name: Yuping Lin

Item Number	Item Name	Price	Quantity	Extended
0905A044	Minced Rib Meat by VX	6.49	80	519.20
2002H136	Soft Monterey Cheese by PY	5.36	80	428.80
1409S039	M8 Peanut Ice Cream	4.5	80	360.0
2910h019	L6 Fruits Cereal	3.33	80	266.40
2910h074	L5 Oat Cereal	4.92	80	393.60

Costs

Total: 1968

Shipping: 0

HST: 256

Grand Total: 2224

Running B2B part on the above three purchase orders, we obtained a procurement order as follow: (we present two screenshots here since the file is too long to fit into one screenshot)

```
Documents                    ✕
  1_cse13122_new.xml
  2_cse13122_new.xml
  3_cse13122_new.xml
  POFiles.xml
  rpt109278076.xml

 1
 2 <procurement id="109278076">
 3     <item number="0905A044">
 4         <code>Order accepted. Confirmation# T2a1700d1445T</code>
 5         <extended>1168.2</extended>
 6         <name>Minced Rib Meat by VX</name>
 7         <ordered>true</ordered>
 8         <price>6.49</price>
 9         <quantity>180</quantity>
10         <wholesaler>Toronto</wholesaler>
11     </item>
12     <item number="0905A112">
13         <code>Order accepted. Confirmation# V2a1700d142dV</code>
14         <extended>842.5195492420736</extended>
15         <name>T6 Sirloin Meat</name>
16         <ordered>true</ordered>
17         <price>5.6167969949471575</price>
18         <quantity>150</quantity>
19         <wholesaler>Vancouver</wholesaler>
20     </item>
21     <item number="0905A123">
22         <code>Order accepted. Confirmation# V2a1700d142eV</code>
23         <extended>298.41787274488524</extended>
24         <name>Fine Beef Meat by PQ</name>
25         <ordered>true</ordered>
26         <price>5.9683574548977045</price>
27         <quantity>50</quantity>
28         <wholesaler>Vancouver</wholesaler>
29     </item>
30     <item number="1409S004">
31         <code>Order accepted. Confirmation# V2a1700d142fV</code>
32         <extended>466.29457523745293</extended>
33         <name>Fudge Ice Cream with Strawberry by AV</name>
34         <ordered>true</ordered>
35         <price>4.6629457523745295</price>
36         <quantity>100</quantity>
37         <wholesaler>Vancouver</wholesaler>
38     </item>
39     <item number="1409S039">
40         <code>Order accepted. Confirmation# T2a1700d1446T</code>
41         <extended>585.0</extended>
42         <name>M8 Peanut Ice Cream</name>
43         <ordered>true</ordered>
44         <price>4.5</price>
45         <quantity>130</quantity>
46         <wholesaler>Toronto</wholesaler>
47     </item>
48     <item number="2002H063">
49         <code>Order accepted. Confirmation# V2a1700d1430V</code>
50         <extended>336.2480498957322</extended>
51         <name>Semi-Cheddar Cheese by JC</name>
52         <ordered>true</ordered>
53         <price>3.362480498957322</price>
54         <quantity>100</quantity>
55         <wholesaler>Vancouver</wholesaler>
```

```
41        <extended>585.0</extended>
42        <name>M8 Peanut Ice Cream</name>
43        <ordered>true</ordered>
44        <price>4.5</price>
45        <quantity>130</quantity>
46        <wholesaler>Toronto</wholesaler>
47    </item>
48    <item number="2002H063">
49        <code>Order accepted. Confirmation# V2a1700d1430V</code>
50        <extended>336.2480498957322</extended>
51        <name>Semi-Cheddar Cheese by JC</name>
52        <ordered>true</ordered>
53        <price>3.362480498957322</price>
54        <quantity>100</quantity>
55        <wholesaler>Vancouver</wholesaler>
56    </item>
57    <item number="2002H123">
58        <code>Order accepted. Confirmation# T2a1700d1447T</code>
59        <extended>576.0</extended>
60        <name>Provolone Cheese by RI</name>
61        <ordered>true</ordered>
62        <price>3.84</price>
63        <quantity>150</quantity>
64        <wholesaler>Toronto</wholesaler>
65    </item>
66    <item number="2002H136">
67        <code>Order accepted. Confirmation# T2a1700d1448T</code>
68        <extended>696.8000000000001</extended>
69        <name>Soft Monterey Cheese by PY</name>
70        <ordered>true</ordered>
71        <price>5.36</price>
72        <quantity>130</quantity>
73        <wholesaler>Toronto</wholesaler>
74    </item>
75    <item number="2910h019">
76        <code>Order accepted. Confirmation# T2a1700d1449T</code>
77        <extended>765.9</extended>
78        <name>L6 Fruits Cereal</name>
79        <ordered>true</ordered>
80        <price>3.33</price>
81        <quantity>230</quantity>
82        <wholesaler>Toronto</wholesaler>
83    </item>
84    <item number="2910h074">
85        <code>Order accepted. Confirmation# V2a1700d1431V</code>
86        <extended>384.32962376536693</extended>
87        <name>L5 Oat Cereal</name>
88        <ordered>true</ordered>
89        <price>4.804120297067087</price>
90        <quantity>80</quantity>
91        <wholesaler>Vancouver</wholesaler>
92    </item>
93    <total>6119.709670885511</total>
94 </procurement>
```

This procurement report shows that our B2B part successfully consolidated the above three purchase orders and procure the items from the best wholesalers.

The Source Code

Source code is attached as PDF file called "sourcecode.pdf" in the root of the submitted zipfile.