|  |
| --- |
| Accelerated Group |
| CD Store Website Readme |
| Term Project Part 1 for CSI 5380 |
|  |
| **Accelerated Group:** |
| **10/15/2013** |

|  |
| --- |
| [Type the abstract of the document here. The abstract is typically a short summary of the contents of the document. Type the abstract of the document here. The abstract is typically a short summary of the contents of the document.] |

# Chapter 1 Environment Setup

In this chapter, the necessary preparation work for Accelerated CD store project will be introduced. The project running environment setting-up server including Java EE installation, Tomcat installation and setup, MySQL database installation and setup, jdbc connector and database connection pool setup, and the IDE setup.

## 1.1 Download and install Java SDK

### 1.1.1 Setup JAVA\_HOME and path to Java SE directory.

My computer🡪properties🡪advanced system settings🡪Environment Variables🡪system variables

Create variable JAVA\_HOME with value c:\java\jdk1.7 (Your java installation directory)

Add values c:\java\jdk1.7\bin (Your java installation directory) to variable path

Create variable classpath and add values %JAVA\_HOME%\lib\dt.jar;%JAVA\_HOME%\lib\tools.jar

### 1.1.2 Download and setup Java EE 1.7.

\*if your OS is Windows 7/8 x64, you have to install it manually in DOS console with parameter –j “your java home/jre”

## 1.2 Install and setup Tomcat web server

### 1.2.1 Download and install Tomcat server

Download tomcat 7.42 from tomcat.apache.org, in our situation, it is installed to c:\tomcat7\ (tomcat home directory)

### 1.2.2 Enable and setup SSL for Tomcat

Most of this part follows the document on CSI 5380 project website By Dr Peyton.

Under DOS console, create keystore file:

Keytool –genkey –keyalg RSA –alias tomcat –keystore my.keystore

Edit server.xml under c:\tomcat7\conf\, add

|  |
| --- |
| <Connector  protocol="HTTP/1.1"  port="8443" maxThreads="200"  scheme="https" secure="true" SSLEnabled="true"  keystoreFile="c:\tomcat7\test.keystore" keystorePass="aaa111"  clientAuth="false" sslProtocol="TLS"/> |

It has to be noticed, if there exists the Apr lib loader, comment it or you have to use openSSL to generate the apr mode keys.

|  |
| --- |
| <!--APR library loader. Documentation at /docs/apr.html  <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />  --> |

In addition, if the glassfish server has been deployed on the machine, the 8080 port might no longer be available to tomcat server. We can use port 80 instead.

## 1.3 Install MySQL and setup connector

### 1.3.1 Download and setup MySQL database

Then download the Database and install it. In this work, MySQL 5.6 is used. User has to create an account in Oracle website. During installation, default port 3306 will be used for MySQL service and the service will be name as MySQL56 by default.

After installation MySQL, a new database cdstore will be created. Database schema and sample data can be imported using cdstore.sql.

|  |
| --- |
| Create database cdstore;  Use cdstore; |

### 1.3.2 Download jdbc connector and setup connection pool.

Download the platform independent file mysql-connector-java-5.1.26.zip from Oracle websit. Unzip it and copy mysql-connector-java-5.1.26.jar into c:\tomcat7\lib\ and add value c:\tomcat7\lib\ mysql-connector-java-5.1.26.jar to environment variable classpath.

As specified in Tomcat document about jdbc connector and MySQL connection pool, following code is added to context.xml under c:\tomcat7\conf\

|  |
| --- |
| <Resource name="jdbc/mysqldb"  auth="Container"  type="javax.sql.DataSource"  factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"  testWhileIdle="true"  testOnBorrow="true"  testOnReturn="false"  validationQuery="SELECT 1"  validationInterval="30000"  timeBetweenEvictionRunsMillis="30000"  maxActive="100"  maxIdle="10"  maxWait="10000"  initialSize="10"  removeAbandonedTimeout="60"  removeAbandoned="true"  logAbandoned="true"  minEvictableIdleTimeMillis="30000"  jmxEnabled="true" jdbcInterceptors="org.apache.tomcat.jdbc.pool.interceptor.ConnectionState;org.apache.tomcat.jdbc.pool.interceptor.StatementFinalizer"  username="root"  password="root"  driverClassName="com.mysql.jdbc.Driver"  url="jdbc:mysql://localhost:3306/CDStore"/> |

Following code is added to web.xml under c:\tomcat7\conf\

|  |
| --- |
| <description>MySQL CD Store</description>  <resource-ref>  <description>DB Connection</description>  <res-ref-name>jdbc/mysqldb</res-ref-name>  <res-type>javax.sql.DataSource</res-type>  <res-auth>Container</res-auth>  </resource-ref> |

It can be also noticed that under each of the project using Tomcat Server, web.xml will be made under WEB-INF directory with above resource reference.

## 1.4 Install and Setup Netbeans IDE:

### 1.4.1 Download and install Netbeans

Netbeans 7.31 can be downloaded from Netbeans.org. In this work, Netbeans is installed to c:\Netbeans\.

### 1.4.2 Setup Tomcat Server in Netbeans

Under menu Tools🡪Servers🡪Add Server, choose Apache Tomcat. Then click next and specify Tomcat installation directory as server location.

# Chapter 2. Design and Implementation

This work is to build a demo CD store website as Part 1 of Course CSI 5380 term project. The requirement of the website has been well defined and the website is designed strictly following what has been illustrated in CSI 5380 lectures.

## 2.1 Architecture of the website

Following the multi-tier MVC design which was taught in class, the whole website can be divided into 4 tiers: Client tier, Web Tier, Business Tier, and EIs Tier as shown in Figure 1. In Web Tier, the Controller collects the requests from the User client (Browser) and decides what kind of action will be made. If the action involves database operations, the controller will use the web service to send the parameters to the Business Tier. The Web services will receive the parameters and start a transaction to connect to MySQL database connection pool. The transaction will be terminated and a result set will be return to the web service. Based on this result set, the web service will generate returned message to the controller and the JSP page would be generated accordingly and be delivered to the customer client.

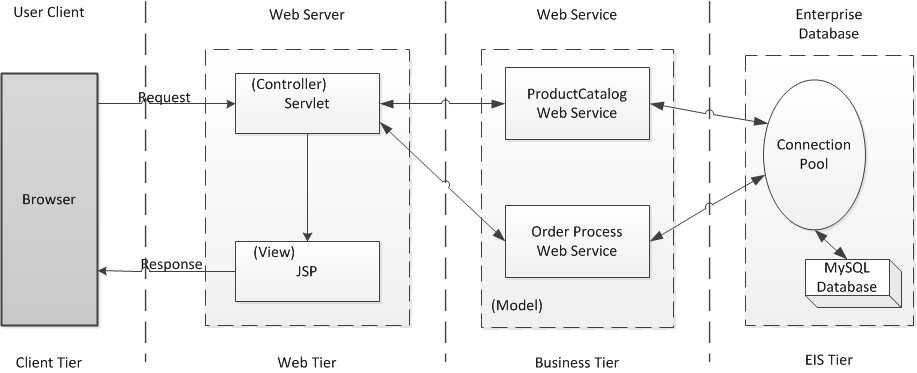


Figure The designed architecture of the website in tiers under MVC mode (Diagram redrawn from the simple MVC Web application architecture diagram in the course note).

## 2.2 Deployment Diagram

The Client tier of the website is just the user’s own browser. The Web Tier, The Business Tier and the EIS Tier are implemented in this work. Figure 2 shows the Deployment Diagram of the CD store website theoretically. The controller part and view part (JSP files) are contained in the Tomcat Web Server. Two web services, the ProductCatalog web service and OderProcess web service are deployed on two different servers. And MySQL database is deployed on the Database Server.

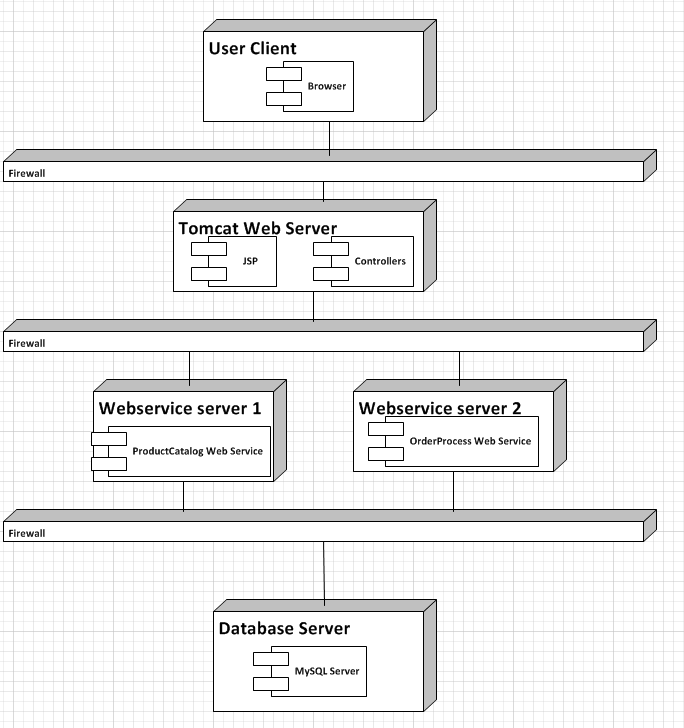


Figure Deployment diagram of the CD store website.

However, due to the nature of the site and the limited physical resource, all these servers, including the Tomcat web server, the web service servers and the database server, are all deployed on the same physical server, which is the developing PC. In the future, these parts can be separated from the main site if required.

## 2.3 Client Tier

In this work, the client tier is typically the user’s received JSP page. Web application framework and .css and .js file are used to define the website style and validating user input. The website template used in this work was downloaded from <http://www.cssmoban.com> . The validator file was generated from <http://www.javascript-coder.com/html-form/javascript-form-validation.phtml> .

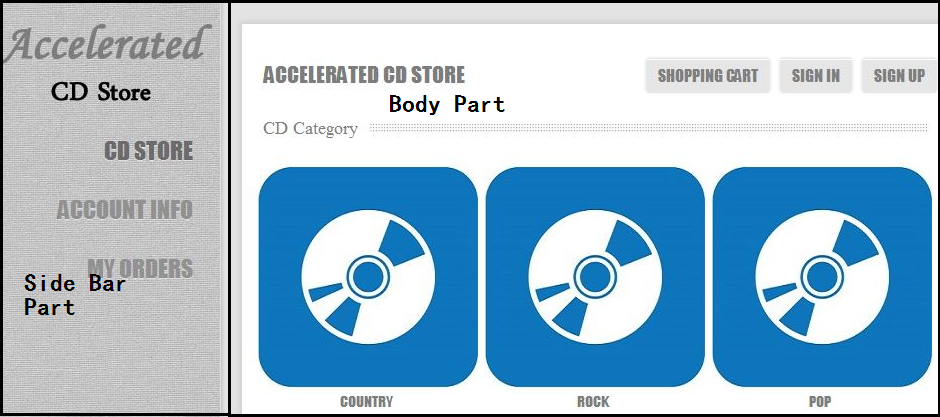


Figure The Side Bar part and the Body part of the JSP pages.

As shown in Figure 3, the GUI of the JSP file can be divided into 2 parts: the side bar part and the body part. The content in Side Part is fixed, providing users the links to front page of the website, to check their account information and to check their order status.

In the Body Part, when any button, link or image is clicked, the controller would capture this operation and find corresponding action in the servlets. If needed, the controller will call a service from two web service servers and transfer the parameters by .xml file. The service server will receive the parameters and conduct the database operation through MySQL database connection pool. The service server will return the result to the web server and JSP pages will be generated accordingly and be displayed in the Body Part in user’s browser.

Later, if needed, this work can be attached with a Banner Part on the top and a Footer Part at bottom of the page. The Banner part can be used to display the logo, some promotion information and some ads. The Footer part can be used to add some legal information, contact information and copy right disclaimer. In this work, these kinds of information is not necessary. Thus these two parts were not added into the design.

## 2.4 Web Tier and Business Tier Source Code Package

According to the design architecture described above, we are having three major parts in this work based on their idea deployment site: the JSP and Controller part on Tomcat Server, the ProductCatalog Web Service on Webservice server 1 and the OrderProcess Web Service on Webservice server 2. We organize our source code into 3 Netbeans projects based on their deployment location.

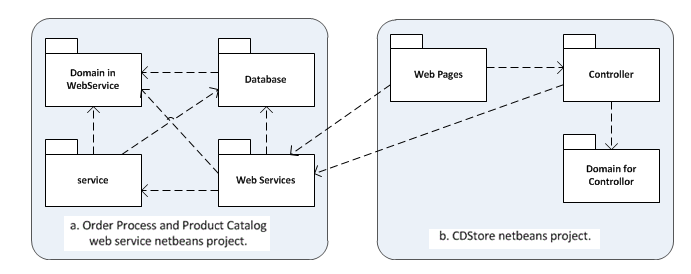


Figure Source code package diagram.

The source code structures of the two web service projects are pretty similar. It contains a domain package defining all the entities need to transfer between the controller and the web services. The database package contains a SQL property file including all the SQL command to be executed, a DBAgent.java servlet defining the basic operations on MySQL connection pool and a Dao.java file defining the functions that the service provided. Dao.java file will call the basic method in DBAgent to have database operations done. The Web Services package is the location stores the web service interface file. The methods that can be called from controller are defined in this package. This package would call the DB operation methods defined in database package. The service package contains two java servlet defining the methods that the web service will provide. This package would also call the DB operation methods defined in database package.

The controller and the JSP pages are contained in CDStore source package. It contains a domain defining all the entities, the Controller package containing the servlets implementing controller functions, the JSP web pages that will be delivered to the user and the Web Service Reference package containing the needed web service. It has to be emphasized that the .css file, .js file and the images used in this work are also included in this package.

## 2.5 Business Process Definition

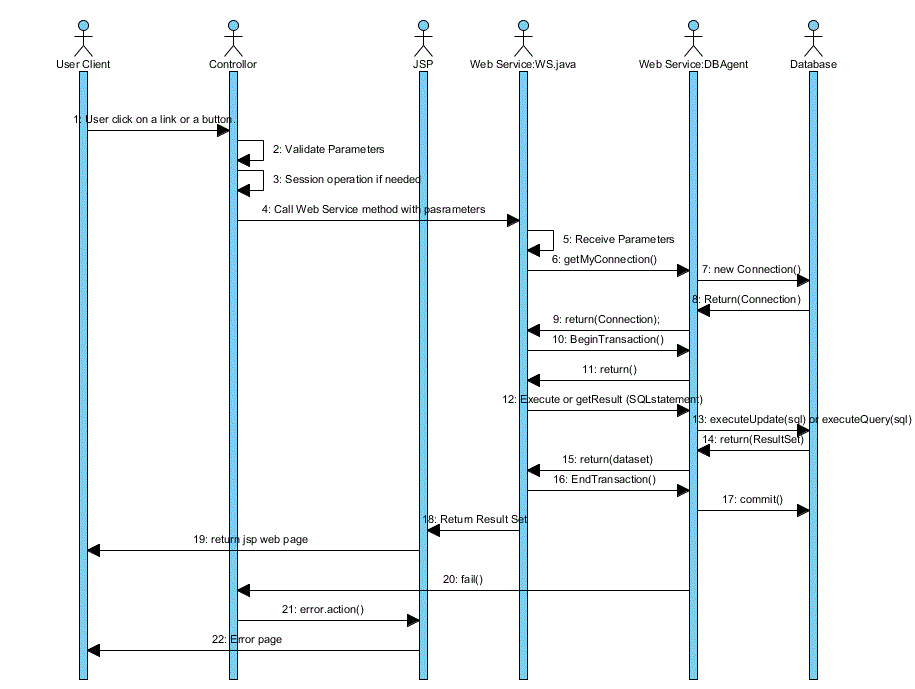


Figure 5. Interaction Diagram of a typical successful user action involing database operation.

Figure 5 represents a set of typical interactions in user, controller, JSP web pages and the web service. This series starts when the user clicks on a link or a button on the web page, or the user trying to open the website page. The controller will then capture this operation and check the validity of the fields needed. Then, if needed, the controller will check session information to see if certain session field has been already obtained. If these validation processes have been passed, the controller will use the web service by sending the service port a generated xml file. After receiving the xml file, the web service gets all the parameters set, get a connection to the DBAgent class via DAO class. DBAgent looks for database connection pool and connects to the database server, then return the connection to DAO. The web service starts a transaction to DBAgent and DBAgent confirms. DAO sends SQL operations to be executed to DBAgent. DBAgent send SQL statement to database and database return the result of execution (result set for query and state for execution). Then DBAgent transfer this result to Webservice. At last DAO sends information to DBAgent to end the transaction and DBAgent send commit message to database. After send the end transaction message, Webservice sends the result set obtained from DBAgent to JSP page. JSP gets the result return to the web server and based on the result, the web page will be generated and delivered to the user’s browser.

In above sequence, the web service WS.java can be replaced with the ProductCatalog web service and the OrderProcess Web Service. The sequence before step 20 represents a successful user operation involving database operation (Query or insertion). No failure would occur in these steps. Step 20 describes a failure from DBAgent is sent to the controller. The controller captures the information and decides to lead JSP page to show user an error page. Any failure on web service WS.java, DBAgent or the database will result in such an error message sent to the controller and lead to an error page delivered from JSP to the user client.

The following user operations are to be defined can be implemented in this project.

|  |
| --- |
| user starts to surf a web page or click link CDStore (view product categories) user clicks a product image or product link (view product information) user clicks the link “Account Info” (try to view account information)  user clicks the link “Add to Shopping Card” (Add an item to shopping cart)  user clicks link “shopping cart” (View shopping cart)  user clicks link “Delete” in Shopping Cart view (Delete an item from the shopping cart)  user clicks the link “check out” (try to check out) user clicks the button “submit” in Account creation page (try to submit the account creation) user clicks the button “sign-in” (attempts to get signed-in)  user clicks link “My account” (attempt to view account information.) |

## 2.6 Database Design

In this work, the following assumptions are made for database scheme implementation:

* 1. Each account only has one shipping address.
  2. Each account only has one registered credit card.
  3. Each customer can only buy one copy of a specific CD product in one order.
  4. Only current account information is kept.
  5. The billing address of the credit card is the same as the shipping address.



Figure ER-Diagram of CDStore website.

From assumption 1 and assumption 2, we can establish a 1-1 relationship between account and its address and between account and its registered credit card. Combined with assumption 4, the address data and credit card data can be integrated with its associated account data. Thus, in our design, the Account entity includes the attributes needed for its address and its credit card. The database scheme is represented as form of ER-Diagram in Figure 6.

# Chapter 3. Implementation and the Source Code

In this chapter, the source code, the implementation and the mapping from design to code will be explained. Also, the contribution of the group members will be introduced.

## 3.1 Database scheme

According to the design in session 2.6, the following database scheme SQL is used in this work. This SQL file is modified from the original database scheme file located on CSI 5380 project web page.

|  |
| --- |
| SET SQL\_MODE="NO\_AUTO\_VALUE\_ON\_ZERO";  DROP TABLE IF EXISTS acount;  CREATE TABLE `account` (  `id` int(11) NOT NULL auto\_increment,  `name` varchar(10) NOT NULL,  `password` varchar(20) NOT NULL,  `fname` varchar(20) NOT NULL,  `lname` varchar(20) NOT NULL,  `payment` varchar(20) NOT NULL,  `street` varchar(50) NOT NULL,  `city` varchar(20) NOT NULL,  `province` varchar(20) NOT NULL,  `zip` varchar(10) NOT NULL,  `phone` varchar(20) NOT NULL,  PRIMARY KEY (`id`)  ) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO\_INCREMENT=7 ;  -- Dumping data for table `account`  INSERT INTO `account` VALUES (1, 'Fan', 'admin', 'Fan', 'Zhao', '1111 2222 3333 4444', 'Tabaret Hall, 75 Laurier Avenue East', 'Ottawa', 'ON', 'K1N 6N5', '12312341234');  INSERT INTO `account` VALUES (6, 'Fan Zhao', 'admin', 'Fan', 'Zhao', '1234123412341234', 'Tabaret Hall, 75 Laurier Avenue East', 'Ottawa', 'Ontario', 'K1N 6N5', '6137907180');  -- Table structure for table `cd`  DROP TABLE IF EXISTS cd;  CREATE TABLE `cd` (  `cdid` varchar(20) NOT NULL,  `title` varchar(60) NOT NULL,  `price` int(10) unsigned NOT NULL,  `category` enum('COUNTRY','POP','ROCK') NOT NULL,  PRIMARY KEY (`cdid`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  -- Dumping data for table `cd`  INSERT INTO `cd` VALUES ('cd001', 'Johnny Cash Greatest Hits', 1599, 'COUNTRY');  INSERT INTO `cd` VALUES ('cd002', 'Willy Nelson Greatest', 1599, 'COUNTRY');  INSERT INTO `cd` VALUES ('cd003', 'Patsy Cline Im So Lonely', 1599, 'COUNTRY');  INSERT INTO `cd` VALUES ('cd004', 'Tragically Hip Fully Completely', 2000, 'ROCK');  INSERT INTO `cd` VALUES ('cd005', 'Beatles White Album', 2000, 'ROCK');  INSERT INTO `cd` VALUES ('cd006', 'Rolling Stones Forty Licks', 2000, 'ROCK');  INSERT INTO `cd` VALUES ('cd007', 'Madonna Greatest Hits', 1799, 'POP');  INSERT INTO `cd` VALUES ('cd008', 'Alannis Morissette Jagged Little Pill', 1799, 'POP');  INSERT INTO `cd` VALUES ('cd009', 'Video Killed the Radio Star', 1799, 'POP');  -- Table structure for table `po`  DROP TABLE IF EXISTS po;  CREATE TABLE `po` (  `id` int(10) NOT NULL auto\_increment,  `status` enum('ORDERED','PROCESSED','DENIED') NOT NULL,  `account` int(11) NOT NULL,  `subtotal` varchar(10) NOT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO\_INCREMENT=7 ;  -- Dumping data for table `po`  INSERT INTO `po` VALUES (4, 'PROCESSED', 1, '2000');  INSERT INTO `po` VALUES (5, 'ORDERED', 1, '3599');  INSERT INTO `po` VALUES (6, 'PROCESSED', 1, '2000');  -- Table structure for table `poitem`  DROP TABLE IF EXISTS poitem;  CREATE TABLE `poitem` (  `id` int(11) NOT NULL,  `orderid` int(11) NOT NULL,  `cdid` varchar(20) NOT NULL,  `price` int(10) unsigned NOT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  -- Dumping data for table `poitem`  -- |

## 3.2 External jars and third-party source code

The following library files are added to netbeans project during the developing process.

--JDK 1.7 (Default): when starting the Java web project, it is added automatically by netbeans.

--Apache Tomcat: when deploying the project to the Tomcat server, it is automatically added by netbeans.

--Metro 2.0: right click on Libraries🡪Add library🡪select metro 2.0 under Global Libraries🡪click “Add library”

--EasyMock 3.2 is used for testing. It is downloaded from easymock.org.

In CDStore project, the style.css file and image files are downloaded from [www.cssmoban.com](http://www.cssmoban.com).

In CDStore project, the gen\_validatorv4.js is downloaded from <http://www.javascript-coder.com/html-form/javascript-form-validation.phtm> .

In the web service projects, part of the source code in DBAgent.java file (Methods: executeSQL, getQueryResult) are from Ezilla (CSI5380Fall2009).

The DB schema file is modified from the original database scheme file located on CSI 5380 project web page.

## 3.3 Source Code Location

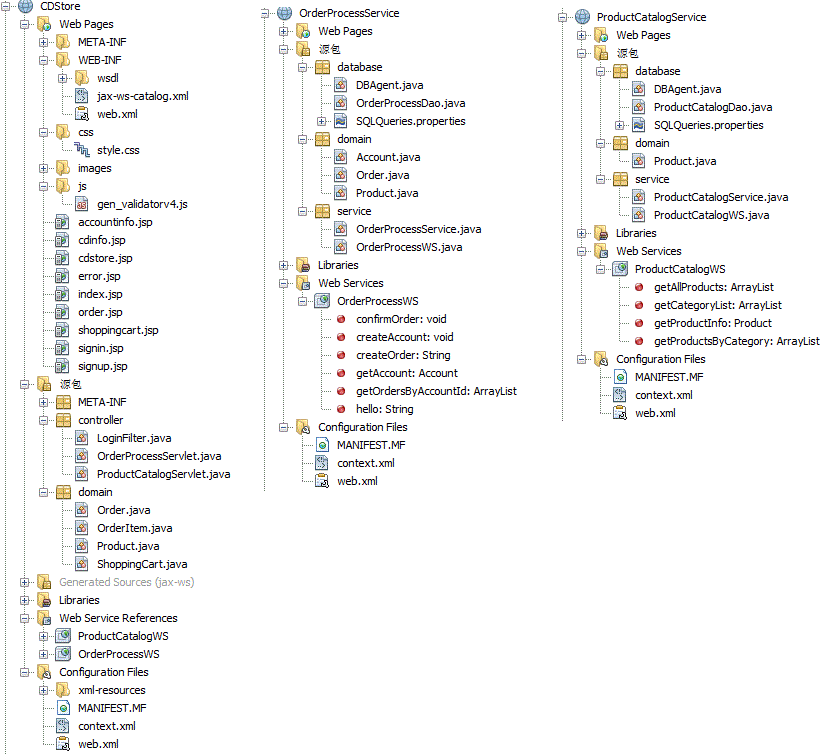


Figure CD store website source code organization.

As illustrated in Session 2.4, there are 3 netbeans projects in this work: CDStore, OderProcessService and ProductProcessService. Their basic structure is shown in Figure 6. The physical location of the source code is as:

|  |
| --- |
| |--CDStore  |--src  |--java  |--controller  loginFilter.java  orderprocessServlet.java  ProductCatalosServlet.java  |--domain  Order.java  Orderitem.java  Product.java  ShoppingCart.java  |--web  |--css  Style.css  |--js  Gen\_validatorv4.js  Accountinfo.jsp  Cdinfo.jsp  Cdstore.jsp  Error.jsp  Index.jsp  Order.jsp  Shoppingcart.jsp  Signin.jsp  Signon.jsp  |--test //This is the location for testing modules.  |--controller  OrderProcessServletTest.java  ProductCataloServletTest.java |
| |--OrderProcessService  |--src  |--conf  MANIFEST.MF  |--java  |--database  DBAgent.java  OrderprocessDAO.java  SQLQueries.properties  |--domain  Order.java  Account.java  Product.java  |--service  OrderProcessService.java  OrderProcessWS.java  |--web  |--META-INF  Context.xml  |--WEB-INF  Web.xml |

|  |
| --- |
| |--OrderProcessService  |--src  |--conf  MANIFEST.MF  |--java  |--database  DBAgent.java  ProductCatalogDAO.java  SQLQueries.properties  |--domain  Product.java  |--service  ProductCatalogService.java  ProductCatalogWS.java  |--web  |--META-INF  Context.xml  |--WEB-INF  Web.xml |

## 3.4 Contribution of Group Members

Fan Zhao: Most of the coding work in java and jsp.

Mingzhe Hu: document in source code

Qiang Wang: readme file except for the test part

Runzhou li: test case and related document

Xiaofeng Wang: test case and related document

Xinyu Zhao: diagrams in readme file

# Chapter 4 Test and Screen Shots

In this chapter, the testing and associate screen shots will be presented. The testing process can be divided into 3 parts: the manual test of the JSP pages and operations on the website, the test module with EasyMock module for servlets, and the test of web services using soapUI.

## 4.1 Manual Test and Screen Shots

As described in session 2.3, only the contents in Body Part in Figure 3 will change upon user’s operation. Thus in this session, to reduce the file size, only the Body part will be presented.

### 4.1.1 ProductCatalog.getCatagoryList

Expectations: When user opens the website, clicking on the link “CD store”, they are expected to see all the categories of the CDs.

Result: The screen shot is exactly same as the one presented in Figure 3. Please go to session 2.3 to see the figure.

### 4.1.2 ProductCatalog.getProductList

Expectations: when the user click on a CD category, the CDs belong to the category will be shown in units with a CD detail link with its title, image and a button that can add it to shopping cart.

Result: The CD information is shown as below.

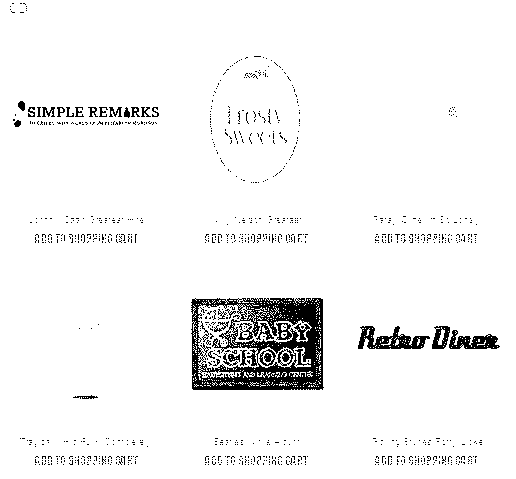


Figure 8. Product information got by product category.

### 4.1.3 ProductCatalog.getProductInfo

Expectations: when the user click on the image of a specific CD, the information of the CD will be displayed.

Result:

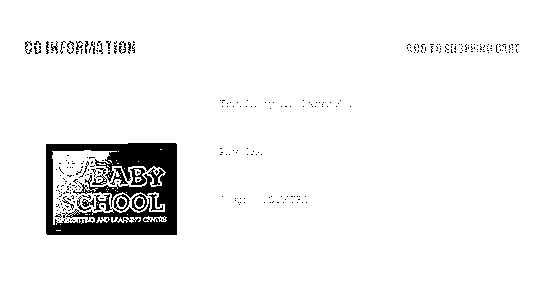


Figure Detailed product information got by product ID.

### 4.1.4 ProductCatalog.invalidIDS

Expectations: when an invalid product ID was input to get product information, an error message will be delivered to the user.

Result:



Figure 10. Invalid Cd ID error message.

### 4.1.5 Unknown actions (for ProductCatalog, OrderProcess and SessionController):

Expectation: an error message will be delivered to user upon an unknown action.

Result:



Figure 11. A return error message when receiving an unknown operation.

### 4.1.6 OrderProcess.createAccount and OrderProcess.getAccount:

Expectation: when user fills all information required, an account can be created. And user can view his account after authenticated.

Result: Creating account process only has a message that account created successfully. However, we can get the information of the account right after.

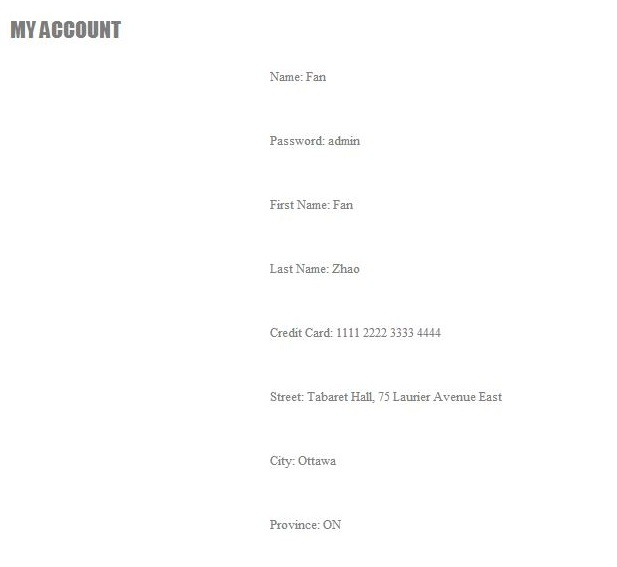


Figure 12. Reviewing of the account information.

### 4.1.7 SessionController.ShoppingCart

Expectation: user can add a CD into shopping cart, remove a CD from shopping cart and can review the item list in shopping cart.

Result: after user adds and removes CD into/from shopping cart, the item list can be reviewed by clicking SHOPPING CART button in Figure 3.

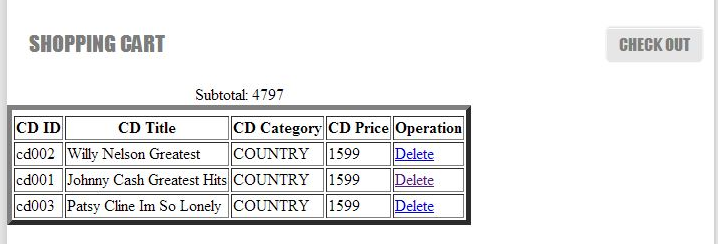


Figure 13. Theitem list in shopping cart.

### 4.1.8 OrderProcess.createOrder, OrderProcess.confirmOrder, and SessionController.reject5th

Expectation: By clicking CHECK OUT button, user can create and confirm an order. And every 5th order will be denied.

Result: user can review the order summary after creation and every 5th order was denied.

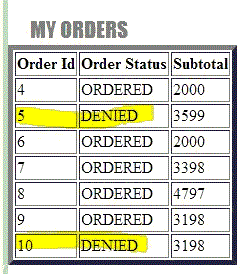


Figure 14. Order status that can be reviewed by the user.

### 4.1.9 SessionController. invalidpassword

Expectation: an error message will be delivered to user if the account and password do not match.

Result:

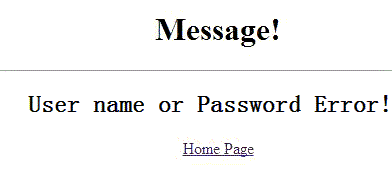


Figure 15. User inputs a wrong username or password.

### 4.1.10 SessionController.missinginfo

Expectation: The user will be noticed when there is any information missing for in wrong format on click submit button for account creation.

Result:

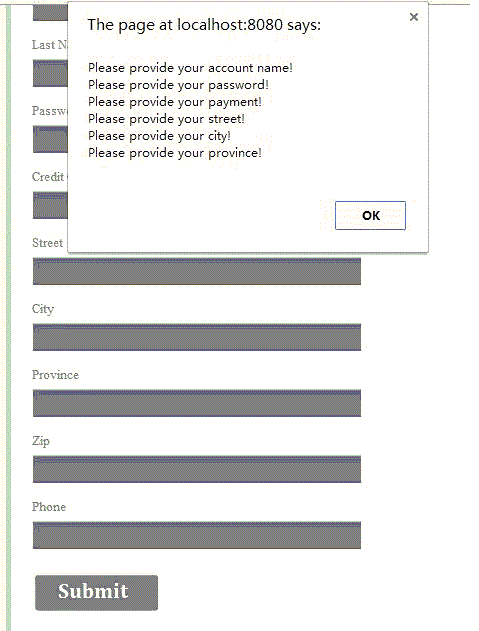


Figure .Notice on missing information when creating an account.

### 4.1.11 Other Tests

The check list also includes some other options such as SissionController.CDStorePage. These options are already integrated into the pages we already tested in above sessions or the test effort cannot be seen by screen shot. These modules and some other modules having been manually tested will be test in unit tests again.

## 4.2 Unit Test

For Controller module, we need to do unit testing. EasyMock provides Mock Objects for interfaces (and objects through the class extension) by generating them on the fly using Java's proxy mechanism. Due to EasyMock's unique style of recording expectations, most refactoring will not affect the Mock Objects. So EasyMock is a perfect fit for Test-Driven Development. After comparing, we decided to use EasyMock as our unit test method to test servlet. Detail test document will be provided upon request.

The CDStore Netbeans project with test unit is like:

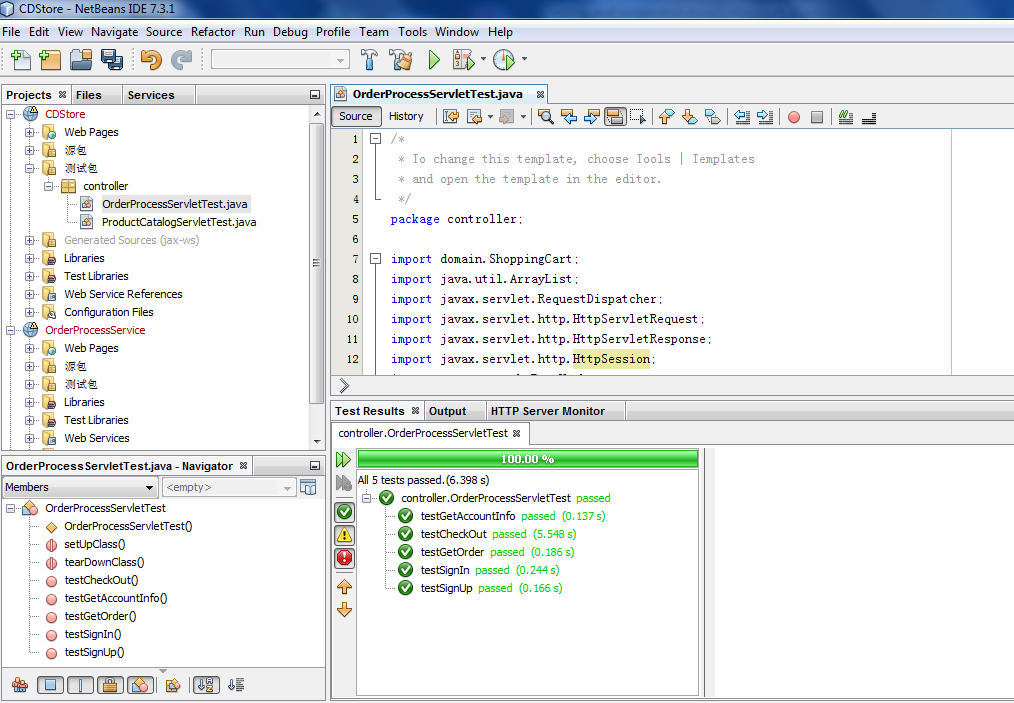


Figure 17. Unit test result for OrderProcessServlet.java.

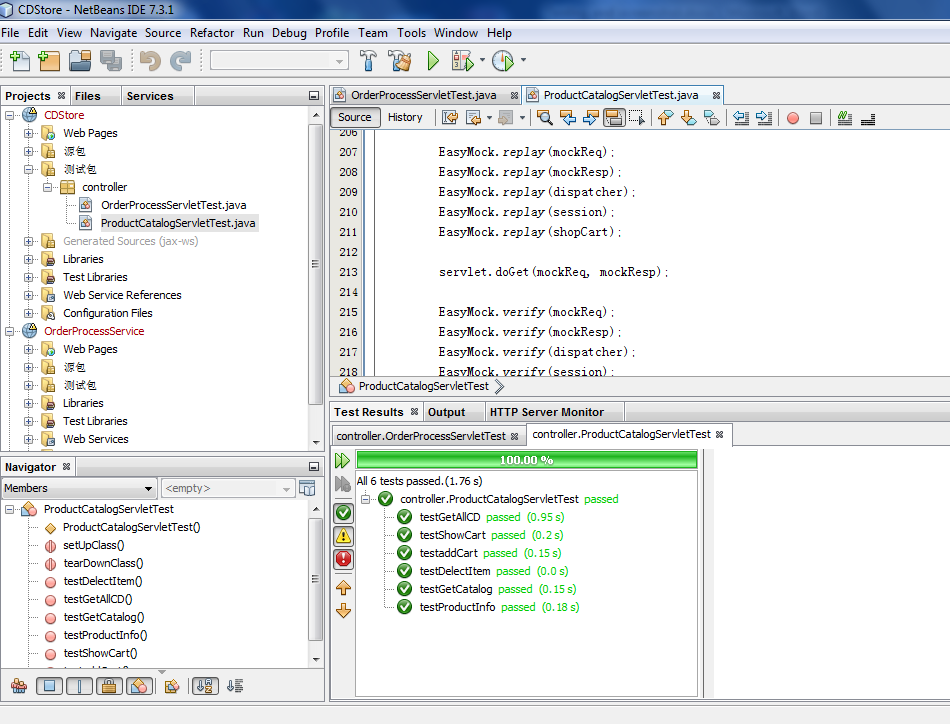
****

Figure .Unit test result for ProductCatalog.java.

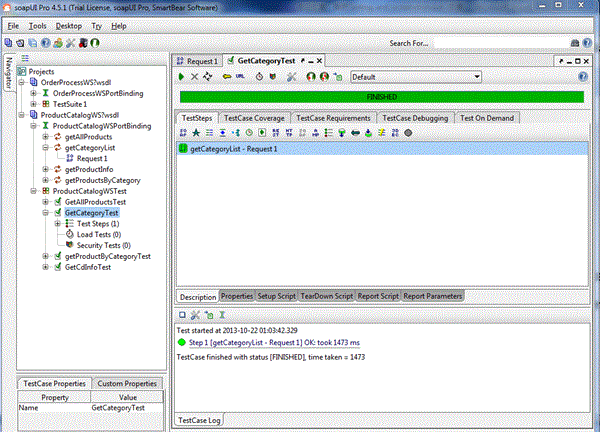
## 4.3 Test for Web Services Using soapUI

For Web Service Testing, our team chooses SoapUI as testing tool. SoapUI is an advanced automated testing tool in wide use. Detail test document will be provided upon request.

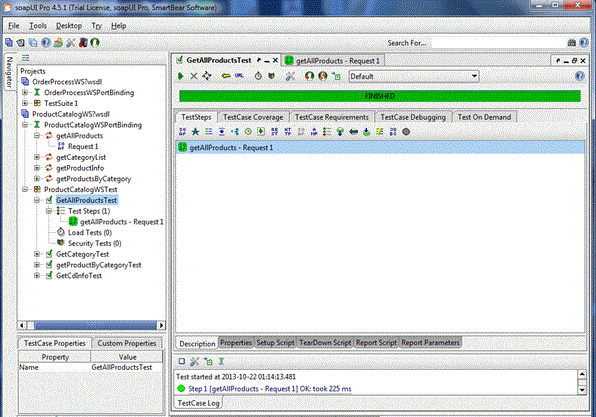
Test Results:

ProductCatalog:

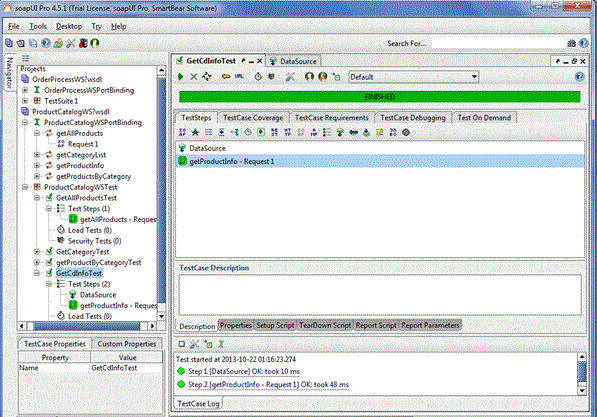
1. ProductCatalog.getCategoryList



1. ProductCatalog.getProductList

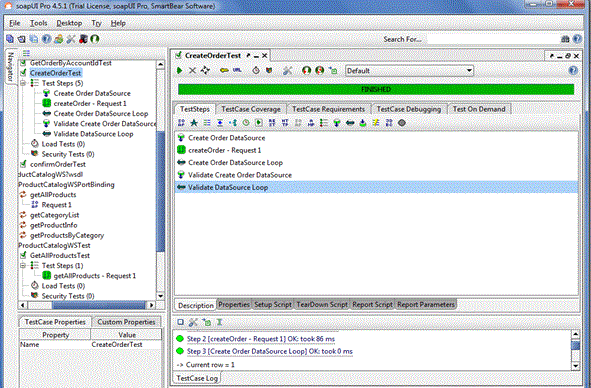


1. ProductCatalog.getProductInfo

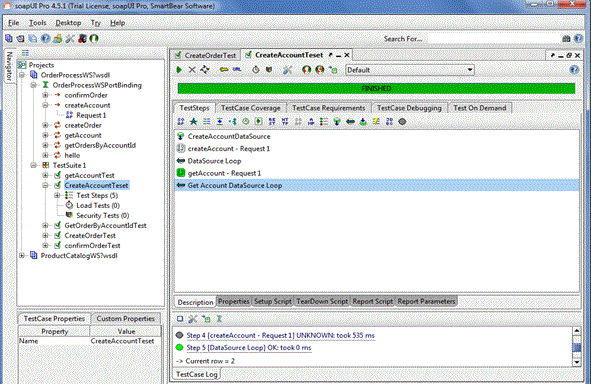


OrderProcess:

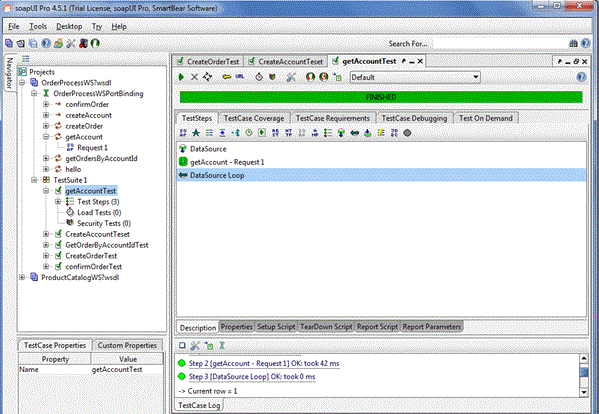
1. OrderProcess.createOrder



1. OrderProcess.createAccount



1. OrderProcess.getAccount



1. OrderProcess.getOrder

