eCommerce CMS Software/Database Design Document

Andrew Won

March 19, 2013

Contents

List of Tables

List of Figures

6 Software/Database Design Document

6.1 Introduction

This document presents the architecture and detailed design for the eCommerce CMS software for the CMSI 402 project at Loyola Marymount University. eCommerce CMS is a software solution for small business customers looking for an initial online presence. The eCommerce CMS application will offer both a web service for HTTP requests and sets of internet-accessible web pages served by that web service.

The eCommerce CMS application will offer a Java-based web service which will allow for HTTP requests to qualified users. The Java-based service will also serve two sets of front-end web pages. Through the first set, small business users of the application can implement and manage an online business management and sales solution. Through the second set, customers of the small business will be able to interact with the small business either through making purchases on an eCommerce web site or through committing information required for generation of a proposal.

6.1.1 System Objectives

The objective of this application is to provide a new content management system with customizable saleable inventory and customer user interfaces for beginning, small-business eCommerce users. Customers will be able to purchase products and various reports and proposals will be able to be generated from a single, easy to customize interface. The small business users of the application will be able to implement and manage an online business management and sales solution. In addition, a secondary objective of this application is to deliver a high degree of customizability requiring relatively low expertise in use of internet technologies.

6.1.2 Hardware, Software, and Human Interfaces

6.1.2.1 *HTTP Requests*

HTTP requests are used by front-end pages to interface with all components in the back-end, see table 1 for a list of these components. The HTTP requests are handled using the following third-party libraries:

(a) JAX-RS Java API that supports web service endpoints according to the Representational State Transfer (REST) architecture. Also known as “Java API for RESTful Web Services.”

(b) Jersey JAX-RS implementation.

(c) JAXB Java architecture for XML Binding.

(d) Jackson Java library for processing the JSON data format.

6.1.2.2 *Keyboard*

A keyboard is used to interface with the user and customer front-ends, see table 1 for a list of these components. Keyboard entries primarily allow customers and users to provide textual data, primarily through form fields.

6.1.2.3 *Pointer Device*

A pointer device is used to interface with the user and customer front-ends, see table 1 for a list of these components. The pointer device, such as a mouse, is the primary way of navigating the user and customer front-ends, which are built primarily using the Menus, Forms, and Dialogs interface style.

6.1.2.4 *Network Interface Controller*

A Network Interface Controller is used to interface with the user and customer front-ends by facilitating a connection for the customer or user to a webpage hosted on a server hosting the application, see table 1 for a list of these components. Because the service is accessed through HTTP requests and the webpages are served through web servers, a network connection is required for all interaction with the application.

6.2 Architectural Design

6.2.1 Major Software Components

6.2.1.1 *Back-End: HTTP Resources*

eCommerce CMS accepts HTTP requests through URI endpoints as outlined later in this document. A user or web browser can initiate the HTTP request to communicate with the web service.

6.2.1.2 *Back-End: Persistent Data Store*

eCommerce CMS offers a persistent data store through a database. The persistent data store is generated by Hibernate, a Java Object/Relational Mapping tool, and stored in an instance of PostgreSQL, a relational database.

The persistent data store maintains saleable items and services, templates for generating documents, templates for generating customer front-end websites, customized customer front-end websites, and histories of transactions.

6.2.1.3 *Back-End: Report/Proposal Generator*

eCommerce CMS offers a report/proposal generator that will aggregate data from the web service's data store. The generator will produce a document with a custom set of fields and a customized layout.

6.2.1.4 *Back-End: Front-End Generator*

eCommerce CMS offers a front-end generator for creating dynamic, highly- customizable customer front-ends. The front-end generator will retrieve customer front-end website properties from the data store and display a customized website to customers.

6.2.1.5 *Back-End: Transaction Processor*

eCommerce CMS offers a transaction processor implemented through a third-party API. The specific API has not yet been determined.

6.2.1.6 *Front-End: What-You-See-Is-What-You-Get (WYSIWYG) Editor*

eCommerce CMS offers a What-You-See-Is-What-You-Get (WYSIWYG) Editor that translates visual changes that a user creates into a set of properties which are then stored in the data store.

6.2.1.7 *Front-End: Inventory Management System*

eCommerce CMS offers an Inventory Management System that allows users to add, edit, and remove saleable items and services from their website. This will be implemented through a form whose data is stored in the data store.

6.2.1.8 *Front-End: Proposal Data Submission System*

eCommerce CMS offers a Proposal Data Submission System. The system allows customers to input data into a form and for users to generate custom proposal documents based on the data entered.

6.2.1.9 *Front-End: Shopping Cart*

eCommerce CMS offers a shopping cart for keeping track of items which are desired to be sold. The shopping cart is implemented through a JavaScript API.

6.2.2 Major Software Interactions

6.2.2.1 *HTTP Endpoint Resources*

The HTTP endpoint resources will interact with both the persistent data store and plain-old Java objects (POJOs).

The HTTP endpoint will interact with the persistent data store through HQL queries processed by Hibernate. The HTTP endpoints will be used as the point of interface for the front-ends and transfer data from the data store to the front-end.

The plain-old Java objects will be used to process data in the persistent data store to be presented to the front-end in meaningful formats.

6.2.2.2 *Report/Proposal Generator*

The Report/Proposal Generator will interact with the HTTP Endpoint Resources and the persistent data store.

The Report/Proposal Generator will communicate directly with HTTP Endpoint Resources to communicate with the persistent data store and to deliver the produced report or proposal to the front-end.

The Report/Proposal Generator will communicate with the persistent data store through the HTTP Endpoint Resources to retrieve the data that will populate the report or proposal. The data that will populate the documents will be determined by the fields selected by the user through the front-end interface.

6.2.2.3 *Front-End Generator*

The Front-End Generator will interact with the HTTP Endpoint Resources and the persistent data store.

The Front-End Generator will interact with the HTTP Endpoint Resources and the persistent data store by retrieving properties of the front-end webpages to generate from the persistent data store and delivering a properly formatted set of front-end webpages to the location where webpages are stored when requested to do so through an HTTP Endpoint Resource.

6.2.2.4 *Transaction Processor*

The Transaction Processor will interact with the HTTP Endpoint Resources, and the persistent data store. Upon the initiation of a purchasing transaction by a customer, the HTTP Endpoint Resource will be used to check the persistent data store to ensure that appropriate availability of the product is available. Upon the completion of the transaction, the HTTP Endpoint Resources will again be used to decrement the inventory appropriately by the number of units sold and to add the transaction to the ItemHistory of the item sold.

6.2.2.5 *What-You-See-Is-What-You-Get (WYSIWYG) Editor*

The What-You-See-Is-What-You-Get (WYSIWYG) Editor will interact with the HTTP Endpoint Resources and the persistent data store. It will also be interacted with through use of the keyboard and a pointing device.

The WYSIWYG Editor will interact with the HTTP Endpoint Resources in order to communicate with the persistent data store. The WYSIWYG Editor will initially, graphically present the customer front-end's present state of properties. When changes are made through the WYSIWYG Editor, the changes will be sent through HTTP Endpoint Resources to be stored in the persistent data store.

Changes in the WYSIWYG Editor will be made through the mouse and keyboard to manipulate visible elements. Manipulation of visual elements will be translated into numeric or textual properties to be stored in the persistent data store.

6.2.2.6 *Inventory Management System*

The Inventory Management System will interact with the HTTP Endpoint Resources, the persistent data store, and with the user through a keyboard. The Inventory Management System will interact with the HTTP Endpoint Resources to communicate with the persistent data store. The Inventory Management System will retrieve data available for items and services and display a summary of this data for users.

Users will be able to interact with the Inventory Management System through keyboard entries into forms, which will be sent to the persistent data store.

6.2.2.7 *Proposal Data Submission System*

The Proposal Data Submission System will interact with the HTTP Endpoint Resources, the persistent data store, and with the customers through a keyboard. The Proposal Data Submission System will interact with the HTTP Endpoint Resources to communicate with the persistent data store. The Proposal Data Submission System will retrieve data available for items and services and display a summary of this data for customers to choose from to build their unique subset of proposal data.

Customer will be able to interact with the Proposal Data Submission System through keyboard entries into forms, which will be sent to the persistent data store to later be retrieved by the Report/Proposal Generator.

6.2.2.8 *Shopping Cart*

The Shopping Cart will interact with the HTTP Endpoint Resources, the persistent data store, the Transaction System, and with the customers through a pointing device.

The shopping cart will locally (on a customer's computer accessing a front-end website) store items that have been selected for purchase by use of a pointing device by a customer. When the customer elects to enter the transaction processor, the shopping cart will determine what data is sent to the persistent data store via the HTTP Endpoint Resources.

6.2.3 Architectural Design Diagrams

Class Diagram.png

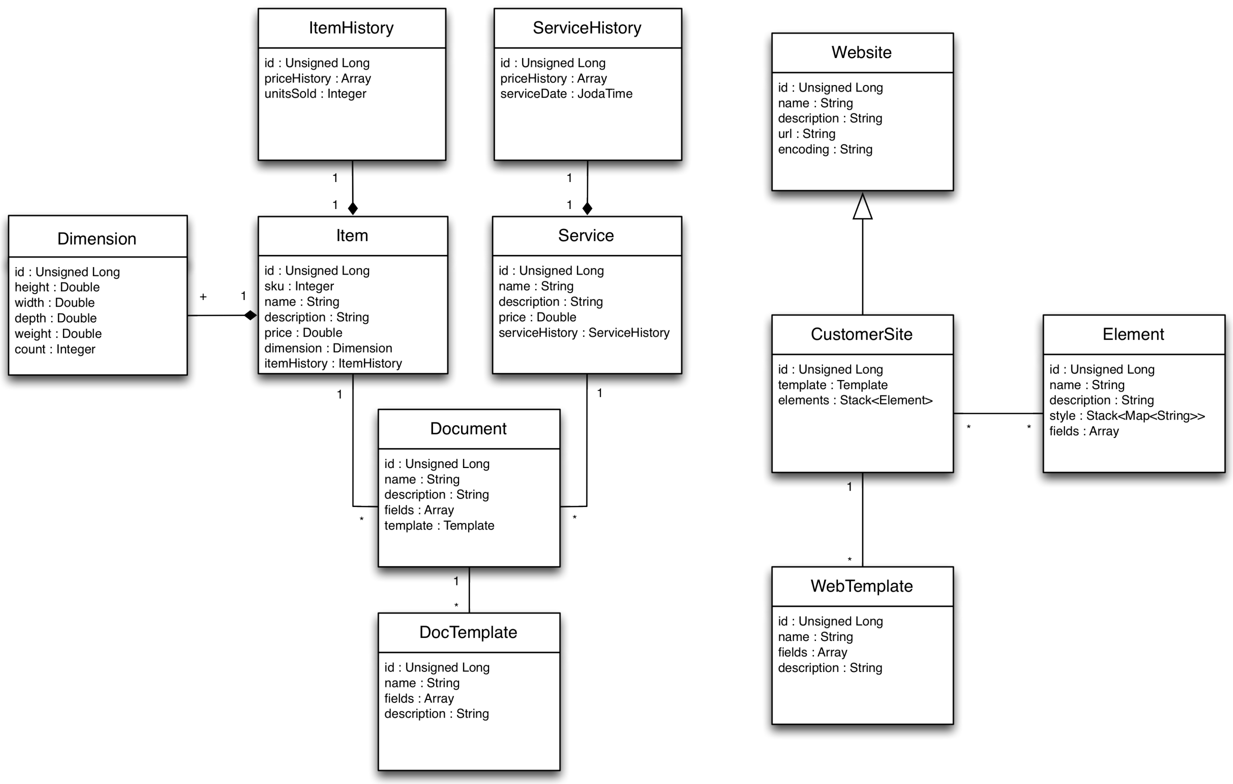


Figure 1: Top-Level Class Diagram

(Client) Diagram.png

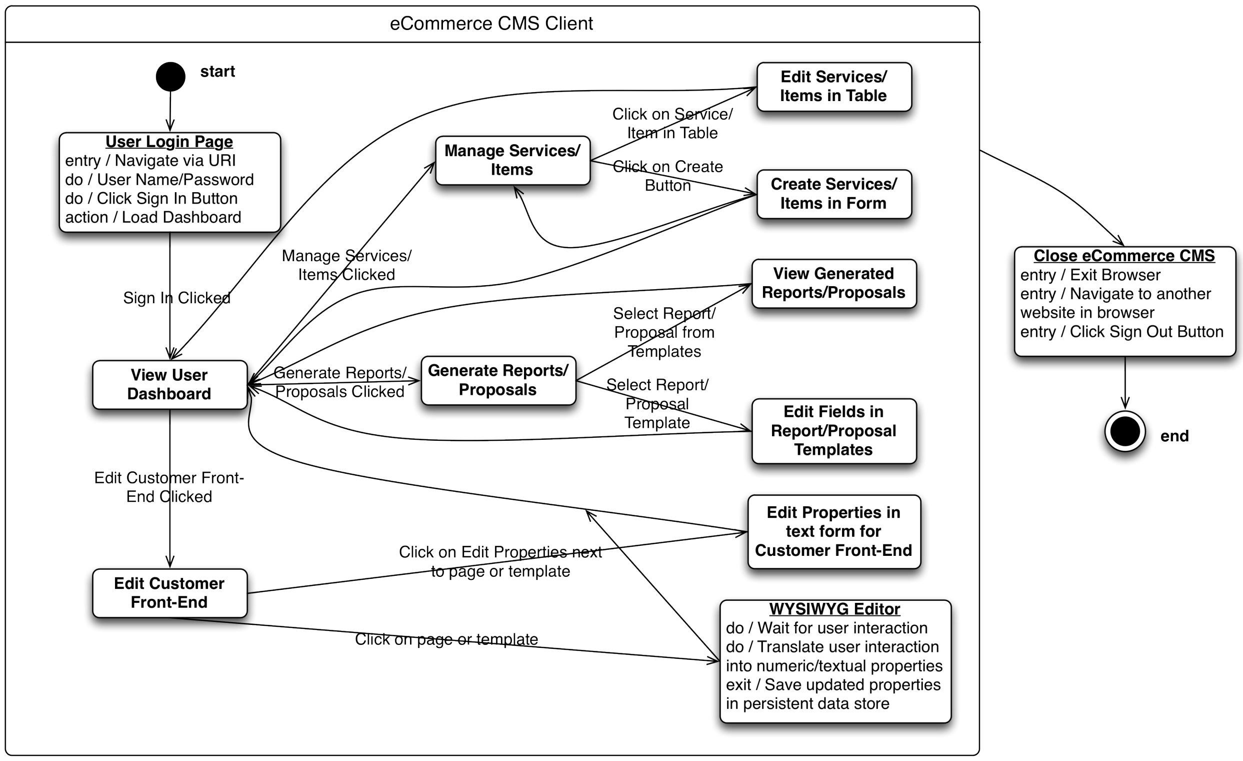

eCommerce CMS Client
start
User Login Page
entry / Navigate via URI
do / User Name/Password
do / Click Sign In Button
action / Load Dashboard
Sign In Clicked
View User Dashboard
Manage Services/ Items Clicked
Manage Services/ Items
Click on Service/ Item in Table
Click on Create Button
Edit Services/ Items in Table
Create Services/ Items in Form
View Generated Reports/Proposals
Generate Reports/ Proposals Clicked
Generate Reports/ Proposals
Select Report/ Proposal from Templates
Select Report/ Proposal Template
Edit Fields in Report/Proposal Templates
Edit Customer Front-End Clicked
Edit Customer Front-End
Click on Edit Properties next to page or template
Edit Properties in text form for Customer Front-End
Click on page or template
WYSIWYG Editor
do / Wait for user interaction
do / Translate user interaction into numeric/textual properties
exit / Save updated properties in persistent data store
Close eCommerce CMS
entry / Exit Browser
entry / Navigate to another website in browser
entry / Click Sign Out Button
end

Figure 2: State Diagram for Client

(Server) Diagram.png

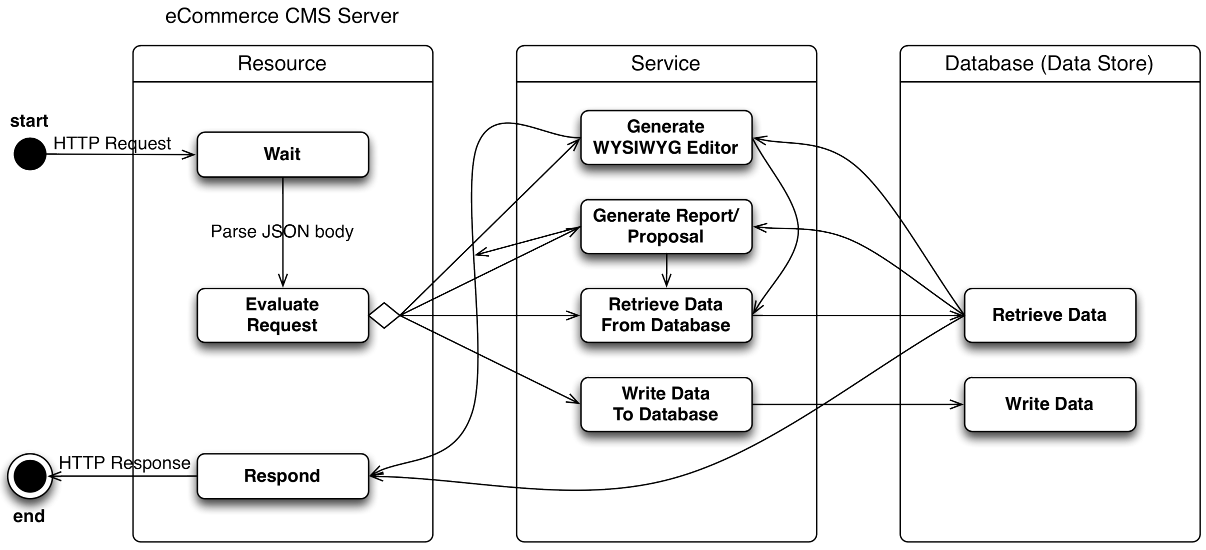


Figure 3: State Diagram for Server

6.3 CSC and CSU Descriptions

The eCommerce CMS application is the Computer Software Configuration Item (CSCI) and consists of three major Computer Software Components (CSC). The CSCI is broken down into the back-end CSC, the user front-end CSC, and the customer front-end CSC. Each respective CSC can further be broken down to several Computer Software Units (CSU). This document will specify some of the CSU's that are intended, as listed in table 1.

CSCI	CSC	CSU
eCommerce CMS	Back-End	Persistent Data Store
	Back-End	Report/Proposal Generator
	Back-End	Front-End Generator
	Back-End	Transaction Processor
	User Front-End	Visual Web Page Editor
	User Front-End	Web Page Template Editor
	User Front-End	Inventory Input/Edit Form
	User Front-End	Report/Proposal Requestor
	Customer Front-End	Proposal Specification Form
	Customer Front-End	Web Store
	Customer Front-End	Purchasing/Transaction System

Table 1: CSCI, CSC, and CSU Hierarchy

6.3.1 Class Descriptions

The CSU's that correspond to the back-end architecture all have corresponding class files. However, not all JavaScript CSU's may have class files because JavaScript does not use classes.

6.3.1.1 *Back-End - Persistent Data Store*

The persistent data store primarily consists of data-access objects (daos) which correspond to equivalent domain object classes. Because these data access objects pertain to various functionality, these classes will be repeated in other sections. The classes are as follows:

- ItemDao.java
- ServiceDao.java
- DocTemplateDao.java
- DocumentDao.java
- CustomerSiteDao.java
- WebTemplate.java

6.3.1.2 *Back-End - Report/Proposal Generator*

The Report/Proposal Generator consists of a resource file to handle HTTP requests, a service file to interface with the database access objects, database access objects to interface with the persistent data store, domain objects to retrieve data to populate the report or proposal, and domain objects to represent the report or proposal as a Java class.

- DocumentResource.java
- DocTemplateResource.java
- DocumentService.java
- DocTemplateService.java
- DocumentDao.java
- DocTemplateDao.java

- ItemDao.java
- ServiceDao.java
- Item.java
- Service.java
- ItemHistory.java
- ServiceHistory.java
- Document.java
- DocTemplate.java

6.3.1.3 *Back-End - Front-End Generator*

The Front-End Generator consists of a resource file to handle HTTP requests, a service file to interface with the database access objects, a database access object to interface with the persistent data store, domain objects to represent the customer website and templates, an interface for websites as a Java class, and utility classes to convert Java objects into HTML, CSS, and JavaScript files.

- SiteGeneratorResource.java
- SiteGeneratorService.java
- SiteGeneratorDao.java
- Website.java (interface)
- CustomerSite.java
- WebTemplate.java
- Element.java
- HtmlCompiler.java
- CssCompiler.java

6.3.1.4 *Back-End - Transaction Processor*

Transaction processing is handled by the Item and Service objects independently, and does not require additional a specific class dedicated to processing transactions. The Item and Service objects will require resource classes to handle HTTP requests and will interface with their respective data access objects with service classes.

- Item.java
- Service.java
- ItemService.java
- ServiceService.java
- ItemDao.java
- ServiceDao.java

6.3.1.5 *User Front-End - Visual Web Page Editor*

The Visual Web Page Editor handles events by a JavaScript file that translates visual changes into numeric and textual properties to be transmitted to the web service.

- wysiwyg.js

6.3.1.6 *User Front-End - Web Page Template Editor*

The Web Page Template Editor looks like the Visual Web Page Editor, but the web page saved into the web service is saved as a template. Templates are not accessible by a web browser outside of the web service and must be implemented into a web page through the Visual Web Page Editor before a customer can access the page.

- wysiwyg.js

6.3.1.7 *User Front-End - Inventory Input/Edit Form*

The Inventory Input/Edit Form will have a JavaScript file to listen for events and communicate with the web service.

- ims.js

6.3.1.8 *User Front-End - Report/Proposal Requestor*

The Report/Proposal Requestor will only have a JavaScript file to communicate with the web service and present a document to the user.

- docRequest.js

6.3.1.9 *Customer Front-End - Proposal Specification Form*

The Proposal Specification Form will have a JavaScript file to submit form data to the web service.

- proposalForm.js

6.3.1.10 *Customer Front-End - Web Store*

The Web Store will have JavaScript files to handle adding or removing items from the shopping cart and to retrieve items/services from the web service and display them on a webpage.

- shoppingCart.js
- store.js

6.3.1.11 *Customer Front-End - Purchasing/Transaction System*

The Purchasing/Transaction System has a file to communicate with the web service.

- checkout.js

6.3.2 Detailed Interface Descriptions

The back-end classes will not directly interact with the user or customer. The back-end is only accessed through the web service by HTTP requests. The following HTTP requests may be made:

Endpoint	Query Parameters	Description
GET /items	page, pagesize, price, name, description, sku	Search for items.
GET /items/:id	–	Get item details.
POST /items	–	Create item.
PUT /items/:id	–	Update item details.
DELETE /items/:id	–	Delete an item.
GET /services	page, pagesize, price, name, description	Search for services.
GET /services/:id	–	Get service details.
POST /services	–	Create a service.
PUT /services/:id	–	Update service details.
DELETE /services/:id	–	Delete a service.
GET /documents	page, pagesize, templateId, field, description	Search for documents.
GET /documents/:id	–	Get document details.
POST /documents	–	Create a document.
PUT /documents/:id	–	Update document details.
DELETE /documents/:id	–	Delete a document.
GET /doctemplates	page, pagesize, name, field, description	Search for doctemplates.
GET /doctemplates/:id	–	Get doctemplate details.
POST /doctemplates	–	Create a doctemplate.
PUT /doctemplates/:id	–	Update doctemplate details.
DELETE /doctemplates/:id	–	Delete a doctemplate.
GET /websites	page, pagesize, templateId, name, description	Search for websites.
GET /websites/:id	–	Get website details.
POST /websites	–	Create a website.
PUT /websites/:id	–	Update website details.
DELETE /websites/:id	–	Delete a website.
GET /webtemplates	page, pagesize, price, name, description	Search for webtemplates.
GET /webtemplates/:id	–	Get webtemplates details.
POST /webtemplates	–	Create a webtemplates.
PUT /webtemplates/:id	–	Update webtemplates details.
DELETE /webtemplates/:id	–	Delete a webtemplates.

Table 2: HTTP Request Web Service Endpoints

6.3.2.1 *User Front-End - Visual Web Page Editor*

The Visual Web Page Editor consists of a webpage that looks similar to an image editor. The Editor will look much like the website the customer will see, but there will be tools surrounding the site to allow the user to customize the site. Elements of the webpage will be laid out on a canvas and a combination of buttons and text input fields will allow users to change the look of the webpage. Text can be directly input into web page elements.

6.3.2.2 *User Front-End - Web Page Template Editor*

The Web Page Template Editor will look exactly like the Visual Web Page Editor. The only difference here is that pages saved as a template will not be accessible to customers until the user goes into the Visual Web Page Editor to implement the generated template into a web page.

6.3.2.3 *User Front-End - Inventory Input/Edit Form*

The Inventory Input/Edit Form will initially present itself as a webpage with a table of existing items/services in the persistent data store. The table of items/services will be directly editable and any changes made in the table will propagate into the persistent data store. A button will be available for an Inventory Input form, and the Inventory Input form will be a form consisting of properties of Item and Service classes.

6.3.2.4 *User Front-End - Report/Proposal Requestor*

The Report/Proposal Requestor will initially present a list of existing templates for reports or proposals. A user will be able to choose from the list and generate a report or proposal that contains data from the Item, Service, ItemHistory, ServiceHistory classes.

The user will also be presented with the option to edit or create a new report or proposal template by editing fields within each document. This will be done through a list of all available fields and a list of included fields. The user will be able to drag-and-drop fields from one list to the other as desired.

6.3.2.5 *Customer Front-End - Proposal Specification Form*

The Proposal Specification Form will look like the web store, but customers will not be able to directly purchase the items or services that are added to a cart. The items or services that are added will be submitted from the customer to the user and the user will have the responsibility of examining the data, preparing a proposal based on the request, and delivering the proposal to the customer.

6.3.2.6 *Customer Front-End - Web Store*

The Web Store will be collection of web pages that are customized by the user. The web pages will contain text, images, products (items or services), and buttons to add products to carts. The Web Store will also have a shopping cart in the top right corner and will link to a “Checkout” page that will handle transactions.

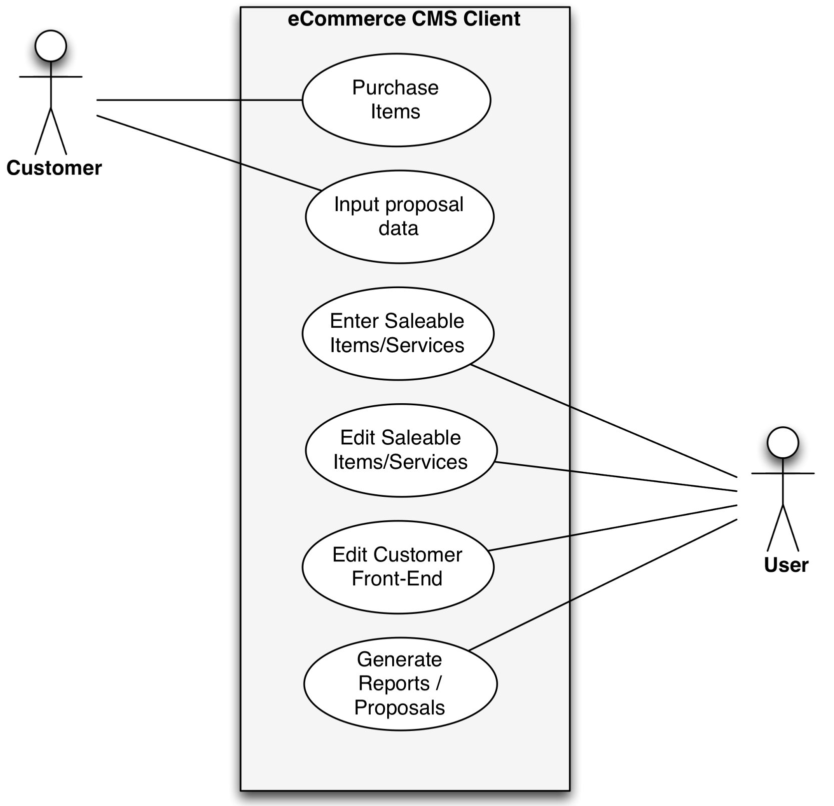
6.3.2.7 *Customer Front-End - Purchasing/Transaction System*

The Purchasing/Transaction System will be a webpage that will display items or services that the customer added to a shopping cart and will use a third-party to execute the purchase/transaction.

6.3.3 Detailed Data Structure Descriptions

All database queries will be parsed into plain-old Java objects (POJOs). The POJOs will contain either java.util.ArrayLists or java.util.Stacks depending upon how the data will have to be accessed.

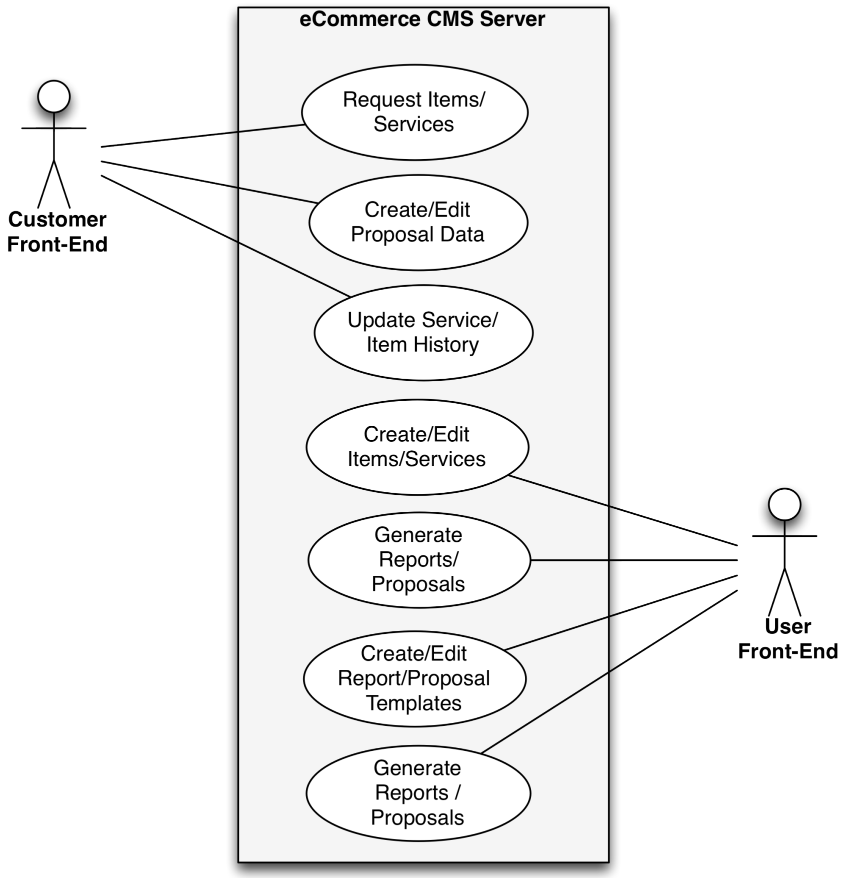
6.3.4 Detailed Design Diagrams



Case (Client) Diagram.png

* Customer indicates a customer of the business implementing the eCommerce site.
** User indicates a user of the eCommerce CMS, typically the business owner, or its agent.

Figure 4: Use-Case Diagram for Client


eCommerce CMS Server
Customer Front-End
Request Items/ Services
Create/Edit Proposal Data
Update Service/ Item History
Create/Edit Items/Services
Generate Reports/ Proposals
Create/Edit Report/Proposal Templates
Generate Reports / Proposals
User Front-End

Case (Server) Diagram.png * Customer indicates a customer of the business implementing the eCommerce site.
** User indicates a user of the eCommerce CMS, typically the business owner, or its agent.

Figure 5: Use-Case Diagram for Server

6.4 Database Design and Description

The persistent data store is implemented with a PostgreSQL relational database and interacts with the Java objects through the Hibernate API. Hibernate designs the database, secures entry into the database, and converts retrieved data directly into plain-old Java objects (POJOs).

The use of Java's Hibernate removes much of the database related work. Additional sections are not added to this document to describe the database because the database is not directly implemented by the programmer.