eCommerce Site
and
Business Management Portal

Software Design Document

***** DRAFT *****

Brian C. Green

Object Oriented Programming Analysis · Final Project

Drexel University, Goodwin College of Professional Studies, winter 2007 Term. Permission is granted to copy, distribute but not modify this document, under terms of academic use.

Document created in:
Philadelphia PA, USA.

This document was prepared using OpenOffice.org 2.2 from Sun Microsystems on a Dell Latitude D400 from Dell Computers Inc. The illustrations where made with Visio 2003 from Microsoft Corporation. The UML illustrations where made using Rational Rose Enterprise Edition, v7.1.9642.27 from IBM Corporation.

THIS PAGE INTENTIONALLY LEFT BLANK

1.1 INTRODUCTION

1.1.1 System Overview

A jewelry retail and manufacturing operation has a need for an application to manage their business. This application should be web-based, and able to accommodate various business functions to handle day-to-day operation. Among the requirements are:

1. Web Based Application
2. Database driven (SQL)
3. Secure
4. Facilitate Retail and Wholesale Operations:
    a. Customer Sales
    b. Wholesale Sales
    c. Process customer payments
    d. Accounts Receivable
    e. Account Processing

2. Facilitate Back Office Operations:
    a. Supply Inventory
    b. Product Inventory
    c. Shipping
    d. Receiving
    e. Accounts Payable
    f. Scheduling

1.1.2 Design Map

Software Methodology

The overall project requirements will be clearly defined in the outset of the project, with dated Gantt chart and clear project responsibilities. Since the customer has no need to review programming technique or progress, a design methodology of the type "Waterfall" will be used. The waterfall methodology allows the project to be developed and managed when segmented into a hierarchy of phases, activities, tasks and steps. This model lends itself well to the Gantt chart, which will help monitor the development process, and increase the likelihood of an on time delivery of the source code.

The development process will be broken into 5 Phases:

Design → Code → Test → Bug Fixes → Documentation

<u>Design Phase</u>

The design phase consists of producing this document, which includes the software design requirements. These requirements will be reviewed by the primary contact for the final deliverable, and once approved, will continue to the "code" phase.

Code

The code phase involves application development, based on the design specifications contained in this document. This phase is further subdivided into three distinct portions: API Interface, Primary Interface, and Database Design.

Test

The testing phase will document all user processes, and verify their functionality. The customer will be involved in this phase to ensure the system is functioning as expected. Additionally, "Bugzilla" will be utilize to track and close any encountered issues.

Bug Fixes

During the testing phase, the bug fix phase will run concurrently, as development repairs any bugs, and testing verifies bug fixes.

Documentation

Finally, in the documentation phase, the final documentation deliverable is prepared and presented to the customer. Documentation will include source, and user documentation, as well as known issues and limitations.

1.1.3 Supporting Material

FEDEX and UPS Integration

The Web API for FedEx and UPS should be utilized for shipping / receiving functions, as well as automatic payment systems. These documents can be found at:

http://www.fedex.com/us/solutions/shipapi/sample_code.html?link=4
http://www.programmableweb.com/api/UPS

Payment systems will be handled using Google's Checkout service to process accounts receivable, payable, and retail checkout. The API can be found here:

https://www.google.com/accounts/ServiceLogin?service=sierra&continue=https%3A%2F%2Fcheckout.google.com%2F%3Fgsessionid%3DnVbqyZXZioQ%26upgrade%3Dtrue&hl=en_US&nui=1&ltmpl=default

THIS PAGE INTENTIONALLY LEFT BLANK

1.2 DESIGN CONSIDERATIONS

1.2.1 Assumptions

Database Architecture

The Database architecture should be SQL based. The customer currently utilizes Microsoft technologies, and requests to maintain the status quo. Thus, Microsoft SQL Server 2005 Enterprise is the preferred database engine.

The hardware required for the database will be determined in testing, based on the performance metrics measured. Hardware requirements will not be considered in the database structure and performance design.

Web Technology

As afore mentioned, Microsoft is the current enterprise technology for the business, and will remain so indefinitely. ASP (Active Server Pages) should be used as the Web Scripting technology, although ASP.NET may also be used if more rapid development is required.

To maintain forward compatibility, VBScript is the scripting language to be used for server side scripting.

Browser Support

The web user interface will support Firefox 1.x and later, and Internet Explorer 6.x and later. Both browsers must be fully tested, and implemented.

API's

The final system will interface with UPS and FedEX for all shipping functions, and Google Checkout for all payment functions.

1.2.2 Constraints

Software

Since VBScript is being utilized, development is restricted to only those functions within the VBScript library. The customer has not authorized the purchase of any libraries, other then those included with VBScript. However, ASP.NET may be utilized if the development group finds it will be beneficial to the customer.

The SQL Server 2005 license will be a 5 concurrent user license. Therefore, the application should be designed to consider the restriction of SQL licensing. Performance monitoring and testing will need to account for, and properly test, in this environment.

Likewise, an existing Microsoft Server 2003 server will be used to house the Web, and Database infrastructure. It is important that the database and web architecture have the ability to be run from the same platform and device.

Hardware

Only one server will be utilized in the final build. Testing and performance monitoring should be baselined on this configuration. The server will have up to two processors, and 2 GB of RAM.

1.2.3 System Environment

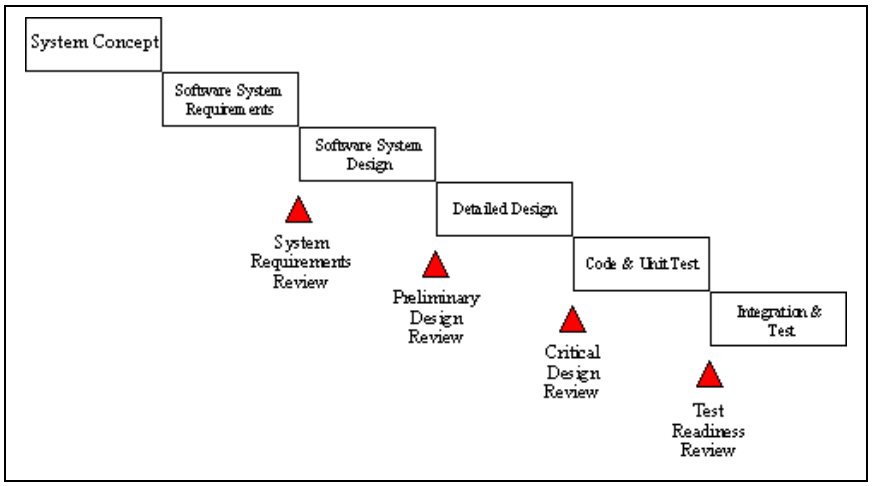
Intel based Microsoft Server 2003 system will be utilized for the final deliverable to the customer. Relevant specifications are:

(2) Intel XEON PIV 3.2GHz Dual Core
(1) 2GB DDR RAM Memory
SATA Raid 1 HD (120GB)

1.2.4 Design Methodology

The waterfall design methodology was chosen for this project, as it seems most appropriate to the design process. A brief description of the Waterfall methodology:

"All projects can be managed better when segmented into a hierarchy of chunks such as phases, stages, activities, tasks and steps. In system development projects, the simplest rendition of this is called the "waterfall" methodology, as shown in the following figure:



In looking at this graphic, which was for major defense systems developments, please note this presumes that the system requirement have already been defined and scrubbed exhaustively, which is probably the most important step towards project success. Nevertheless, the graphic illustrates a few critical principles of a good methodology:

- *Work is done in stages,*
- *Content reviews are conducted between stages, and*
- *Reviews represent quality gates and decision points for continuing.*

The waterfall provides an orderly sequence of development steps and helps ensure the adequacy of documentation and design reviews to ensure the quality, reliability, and maintainability of the developed software. While almost everyone these days disparages the "waterfall methodology" as being needlessly slow and cumbersome, it does illustrate a few sound principles of life cycle development."

This design methodology will be utilized for a phased approach to design. No phase can begin, until its master has been completed, with the exception of Testing and Bug Fix, which can run concurrently.

1.2.5 Risks

Google API Risk

The Google API, which was chosen for the checkout system, was chosen due to its low initial cost and current usage trends. However, this interface is relatively new, and in Beta testing. This indicates it is possible for the API interface to change as well, which may require follow-up, or redesign of the interface prior to the project completion.

1.3 ARCHITECTURE

The software will consist of a database, an Operating System, a web server, VB Code, and several APIs to third parties:

Operating System:	MS Windows 2003
Database:	MS SQL Server 2005 Enterprise
Web Server:	MS Internet Information Services 6.0
Code:	ASP VBScript
API:	Google Checkout
API:	FedEX
API:	UPS

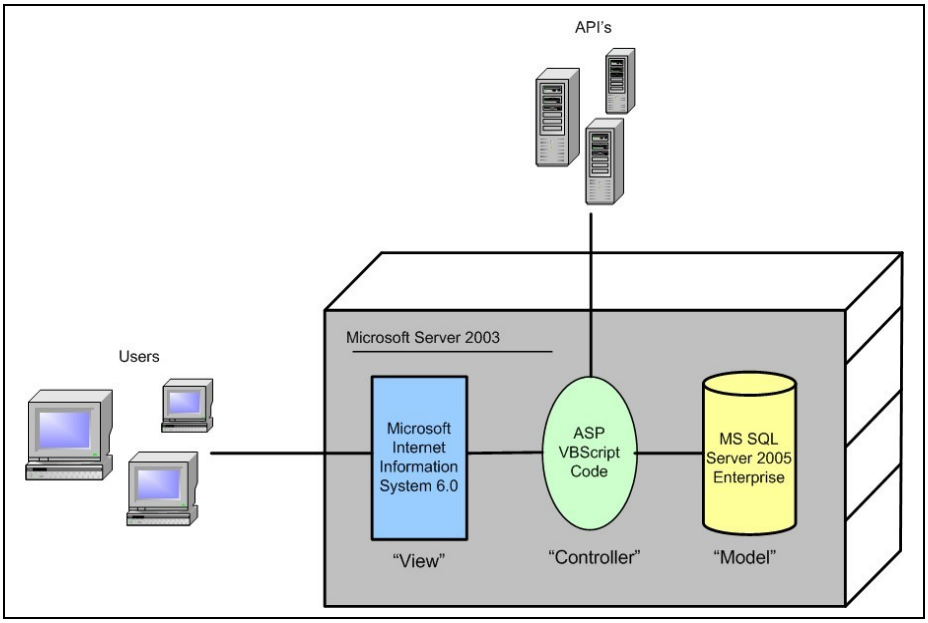


Figure 1.3.0.1

The high-level interaction of the components can be seen in figure 1.3.0.1. Using the Model-View-Controller method, users interact with a web interface, driven by the code on the server, which stores the data in the Microsoft SQL Server Database. This architecture allows for independent development of each component.

1.4 HIGH LEVEL DESIGN

1.4.1 Use Case

There are three possible user roles:

Guest

This is a visitor without an account. This visitor can browse the "store", and request information. Additionally, this user can create an account. However, until the account is created, this user can not purchase products, or make payments.

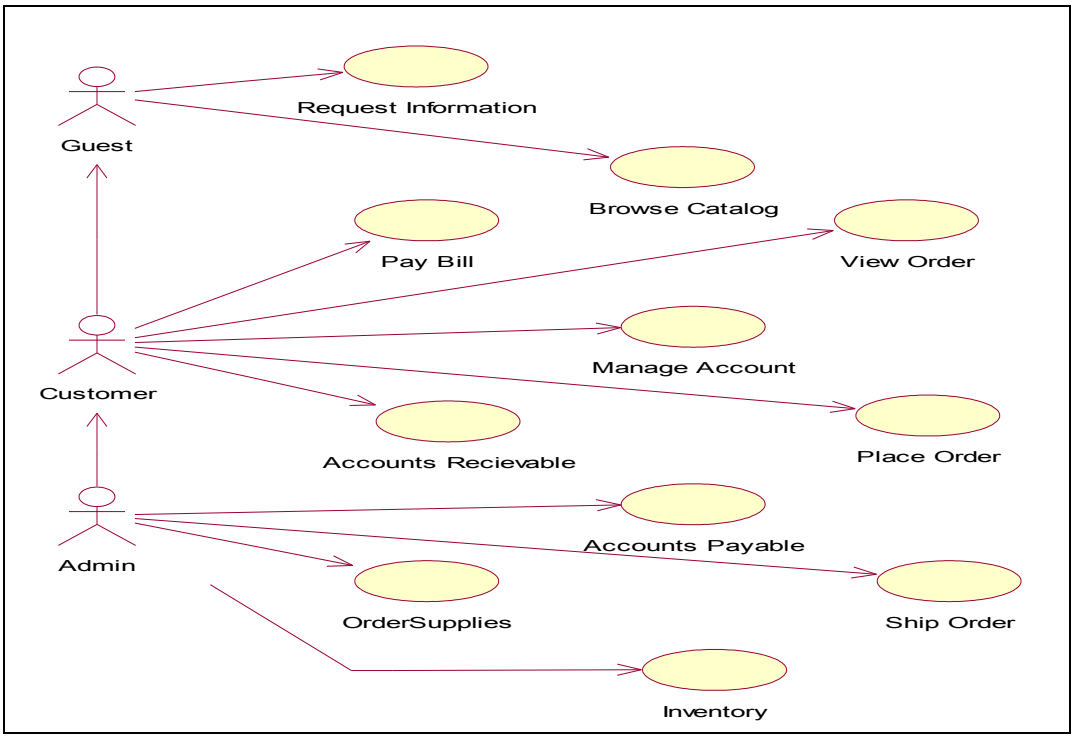
Registered User

This user inherits all the functionality of a guest, but can also order goods, check order status, make payments, and change their preferences.

Administrator

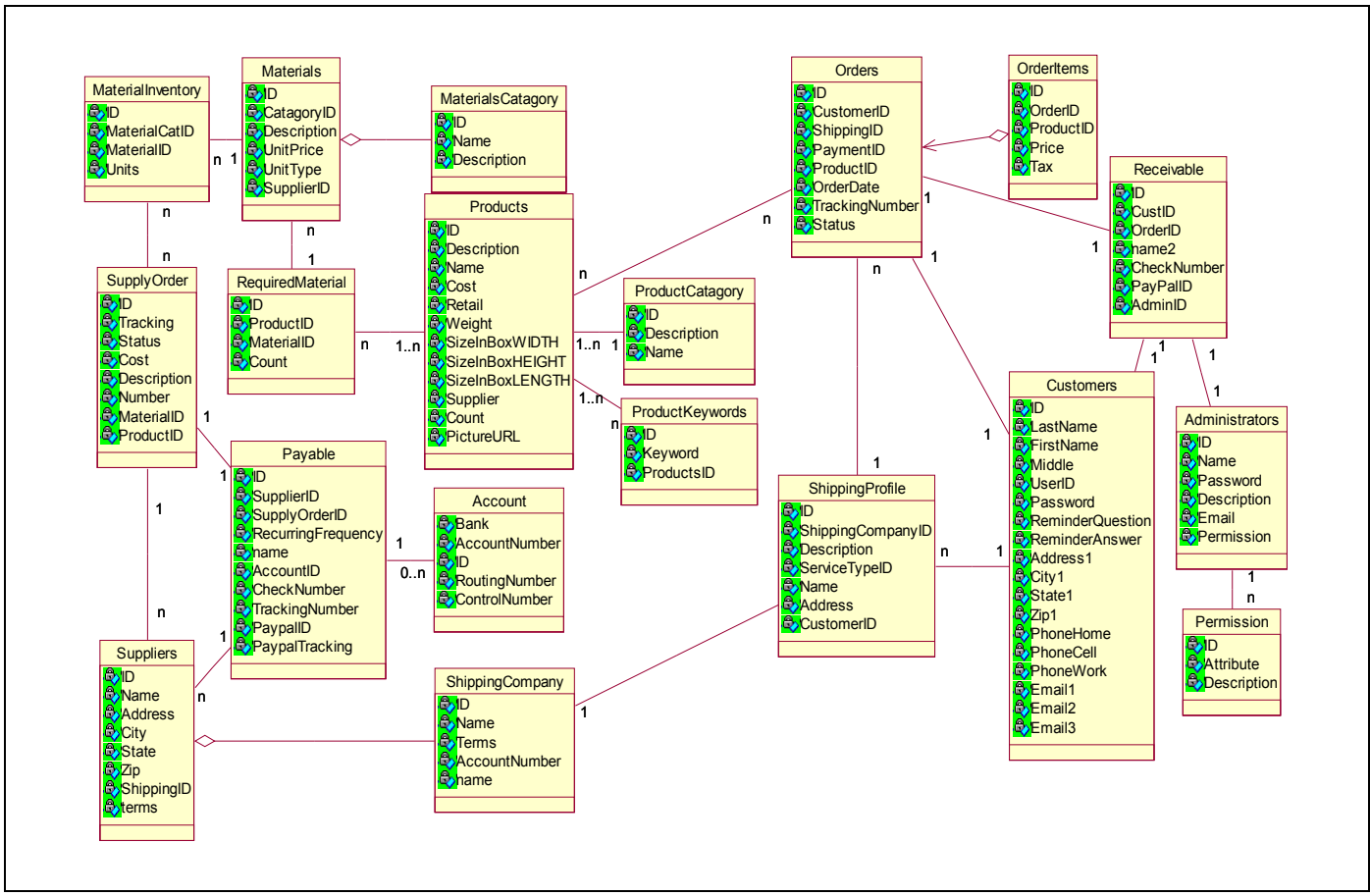
Administration users inherit all functions from the registered user, but can also change application settings, manage the store front, order supplies, perform inventory functions, and accounting.

The use case is as follows:



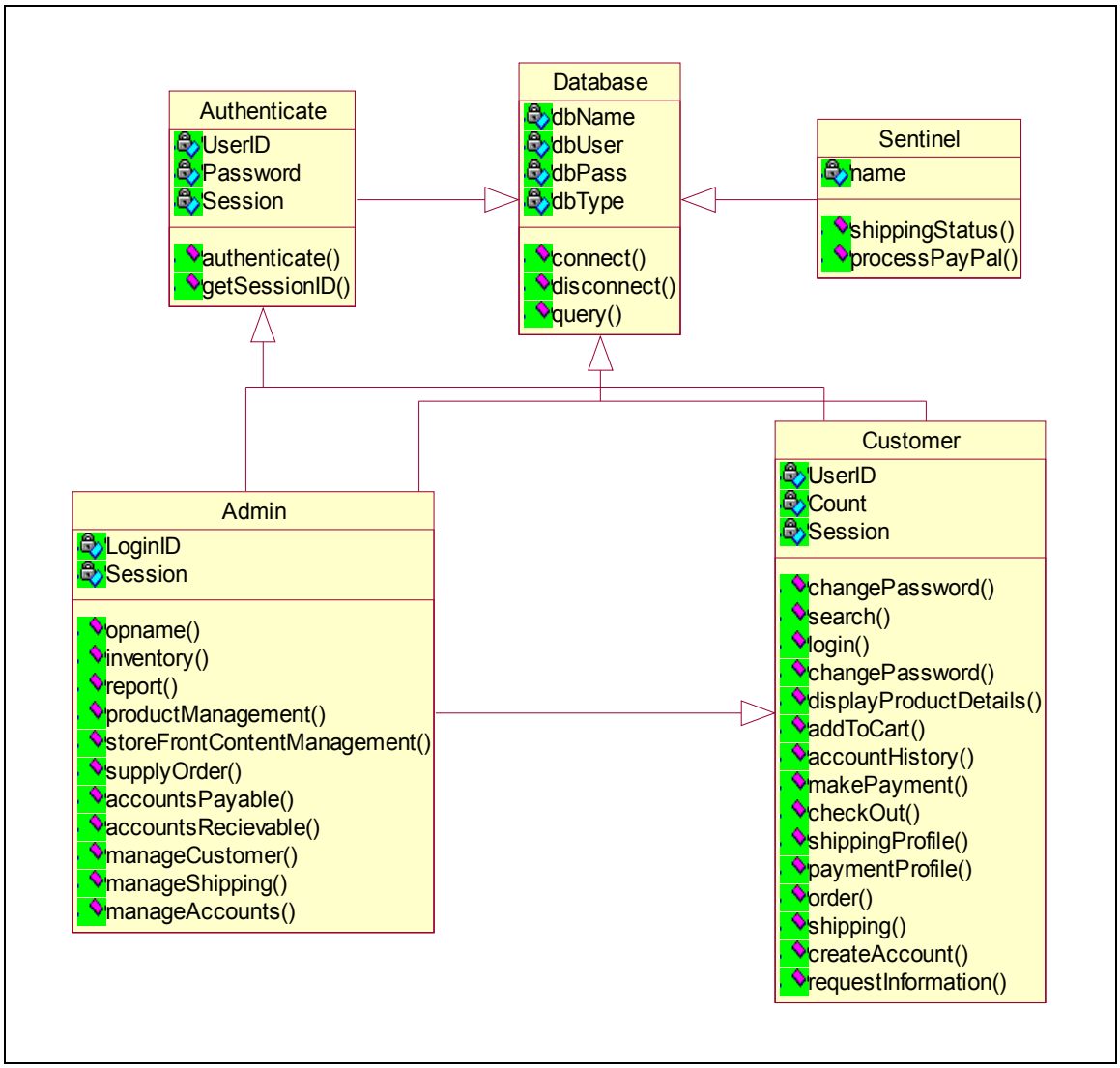
1.4.2 Database Design

Database Class Diagram:



1.4.3 Web Server Design

Web Class Diagram:

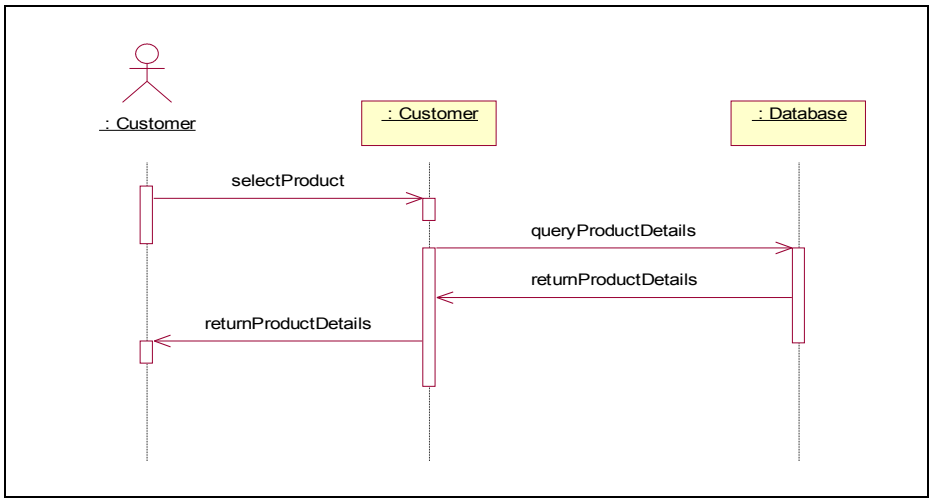


NOTE: In the Web Class Diagram, the classes represent the control elements. These classes are each created from a View class (not shown).

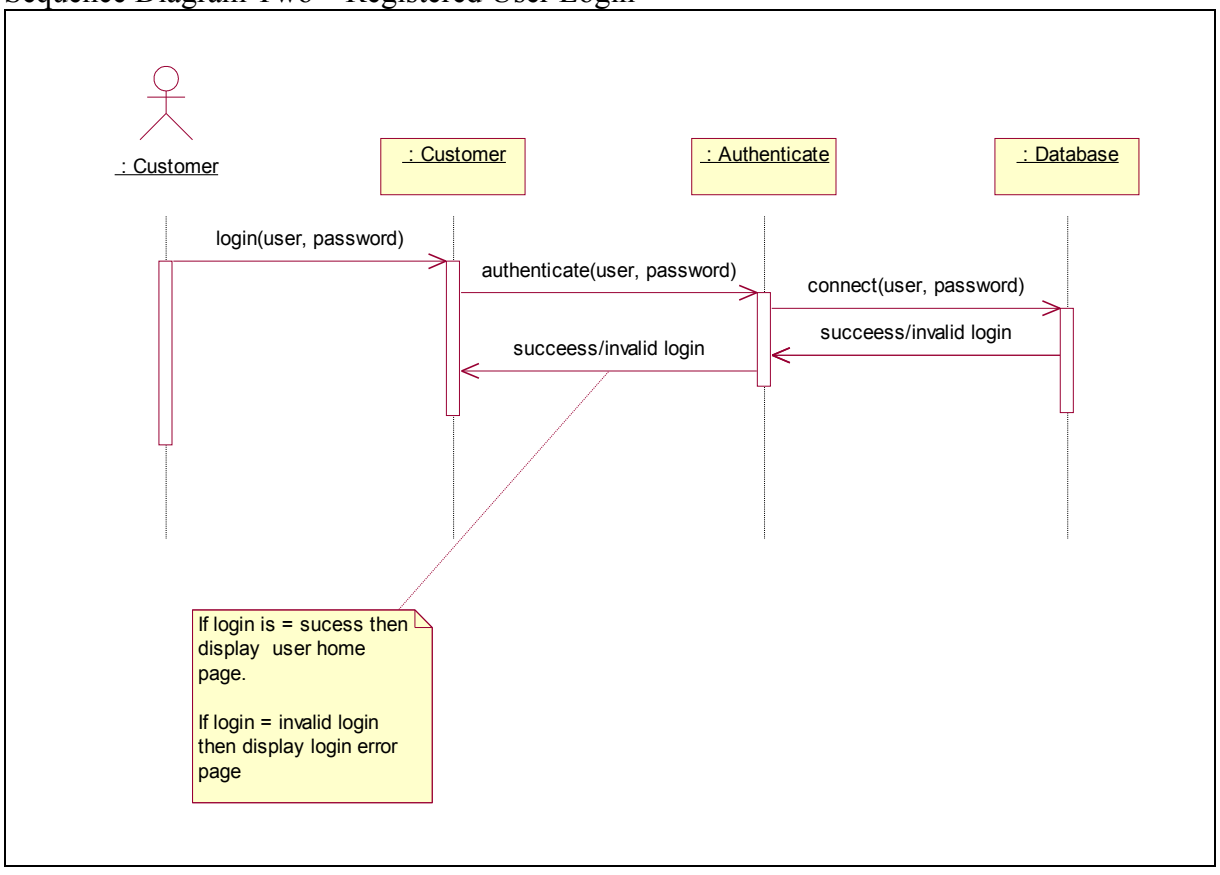
1.5 LOW LEVEL DESIGN

1.5.1 Sequence

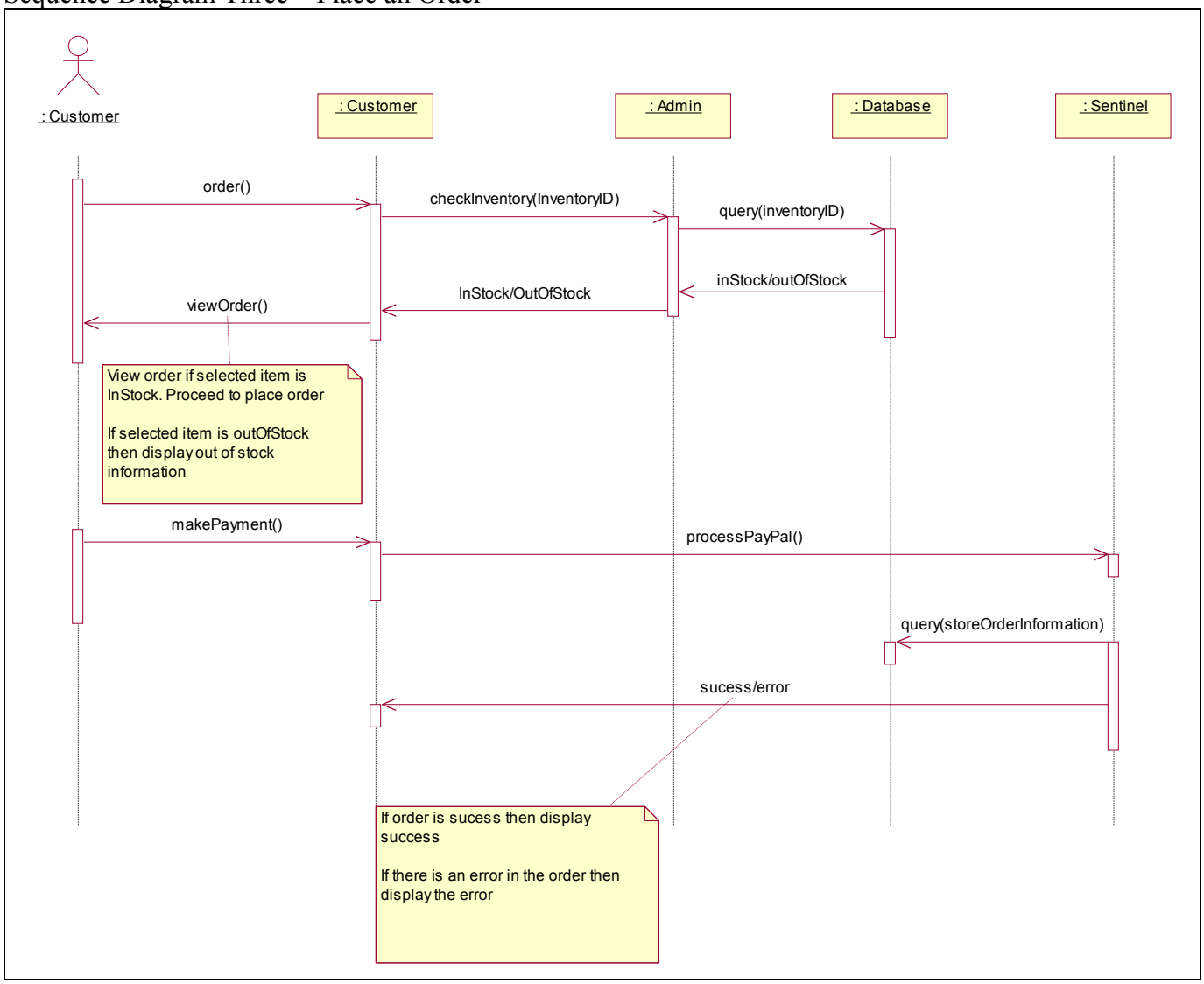
Sequence Diagram One – Browse Catalog



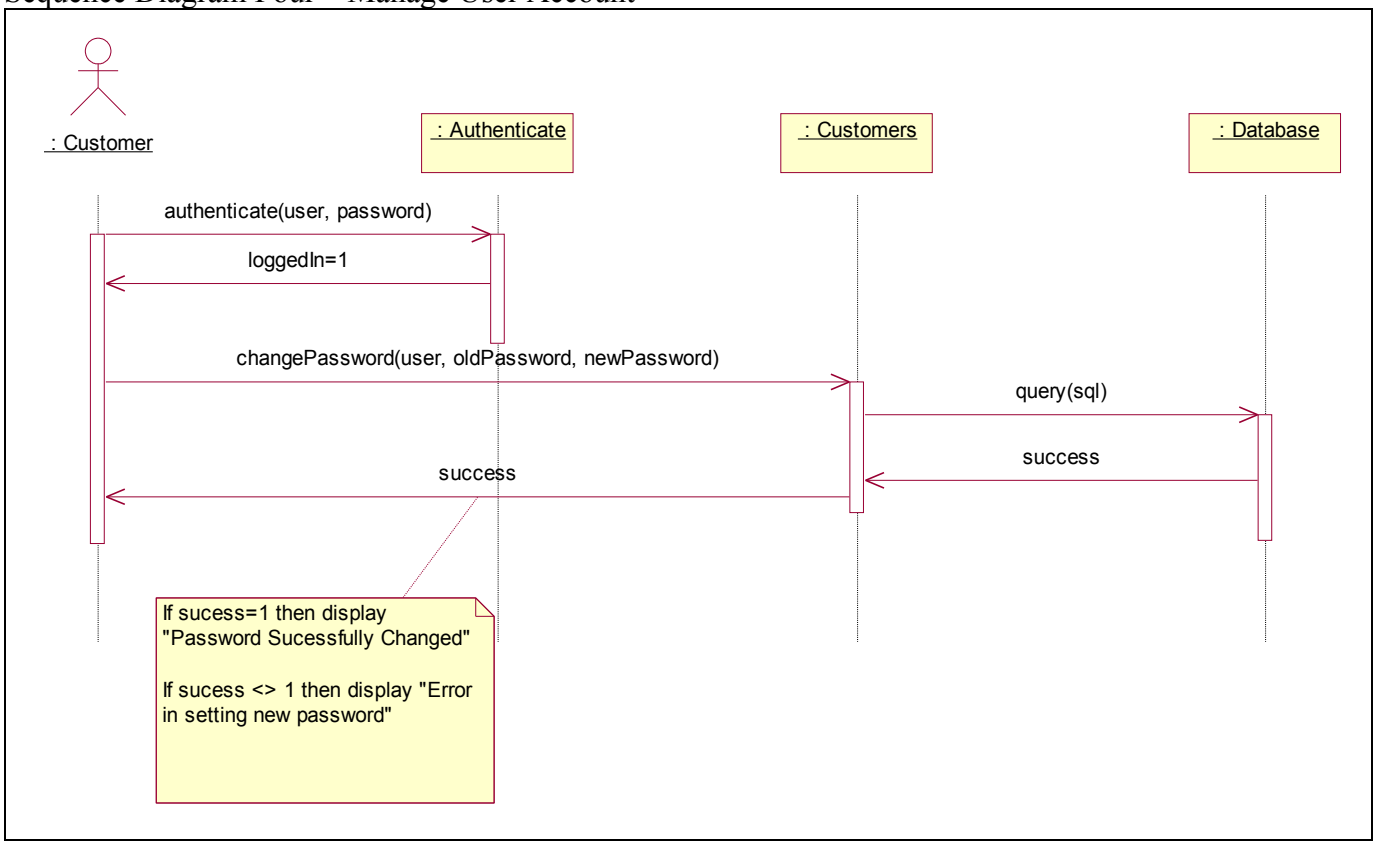
Sequence Diagram Two – Registered User Login



Sequence Diagram Three – Place an Order



Sequence Diagram Four – Manage User Account



1.5.2 Function List

Function Definition Conventions:

functionName(argument, [optional])
Brief description of the function
→ Input Arguments
Brief description of each argument
← Return
Brief description of return variables

Database Class Functions

connect(server, user, pass)
Connects to the specified database.
→ Server, User, Pass
Specify the Server name (i.e., (local)), a valid database user, and the associated password.
← ID
If the connection is successful, the connection ID will be returned. Failed connections return false.

disconnect(ID)
Closes the database connection to the specified connection ID.
→ ID
Specifiy the connection ID.
← 1/0
Will return true if the connection is closed, else returns false.

query(ID, query)
Executes the requested query on the specified connection ID.
→ ID, query
Specify the connection ID. Provide a valid SQL Statement
← b, ra, result

"b" will return true if no errors encountered from SQL, false if an error was encountered.

"ra" is the number of rows effected by the SQL statement, if returned by DBMS

"result" is the record set, as an associative and numerical array. Associations are the column names.

Sentinel Class Functions

shippingStatus(provider, tracking, [account], [receiver])

Handles all shipping functions, and shipping API.

→ provider, tracking, account, receiver, action

- Provider must be equal to FedEX or UPS.
- Tracking – tracking number. If 0, then shipping function is called, using the account, and receiver variables.
- Account – provide only if tracking is set to "0"Receiver – an array – Lastname, Firstname, Address, City, State, Zip, Country, Apartment, Special. Must be a numerical array.

← status

Returns a tracking number if a shipment is successfully placed. If tracking, the status returns true if delivered, and 0 if NOT delivered.

processPayPal(ID, amount, payment)

Handles Google Checkout functions.

→ ID, user, pass, amount, payment

- ID - Provide payment ID.
- Amount – Payment amount. Positive if a charge, negative for a refund or void.
- Payment – Array containing CardID, Expiry, Type, NameOnCard, and CardCode

← ID

Returns the transaction ID. Returns false if the payment fails.

Authenticate Class

authenticate(ID, pass)

Authenticates a user using the ID and password.

→ ID, pass

- ID – User login ID.
- Pass – Password

← ID

Returns false if login fails, else the session ID.

session(ID)

Check a session.

→ ID

ID – the session ID.

← ID

Returns status of session, true or false.

Admin Class Functions

inventory(ID, [count])

Handles inventory functions.

→ action, ID, count

- ID – The product ID.
- Count – If a count is entered, the product inventory count will be changed. Use +n to increment, and –n to decrement the count. An integer with no operator will change the inventory count to the supplied integer.

← count

The items inventory count after function is run.

report(type, array)

Provides report.

→ type, array

- Type – specify the type of report.
- Array – provide the report variables as an associative array.
- ***Report function is documented in accounting design documentation.

← report

The report text is returned, as an array. Position zero is the number of columns per row.

productManagement(ID, [detail])

Change product description, and image.

→ ID, Detail

- ID – The product ID.
- Detail – Array containing: Name, Description, price, packageingDimensions, and ImageURL

← detail

Product details, as array containing: Name, Description, price, packageingDimensions, and ImageURL.

storeFrontContentManagement(ID, status)

Adds and removes products from the store.

→ ID, status

- ID – The product ID.
- Status – 1 = show product, 0 = hide product

← 1/0

Returns false if error, else returns true.

supplyOrder(ID, [count])

Changes status of non-revenue items.

→ ID, count

- ID – The material ID.
- Count – If a count is entered, the inventory count will be changed. Use +n to increment, and –n to decrement the count. An integer with no operator will change the inventory count to the supplied integer.

← count

Returns the count on the material ID

accountsPayable(ID, number, amount)

Updates supplier account details with payments owed and sent.

→ ID, number, amount

- ID – Account ID.
- number - check or invoice number Prefix checks with "-" and invoices with "+".
- Amount – invoice or check amount

← id

Returns the accounting package control number. (ID field in database).

accountsRecievable(ID, number, amount)

Updates charge accounts with payments received.

→ ID, number, amount

- ID – Account ID.
- number - check or invoice number Prefix checks with "-" and invoices with "+".
- Amount – invoice or check amount

← id

Returns the accounting package control number. (ID field in database).

manageCustomer(ID, [details])

Updates accounts.

→ ID, details

- ID – Account ID.
- Details – "0" to convert to non-charge account, "1" to "99" to convert to a charge account, where the integer is the number of days the net is due.

← details

Returns the terms, 0 – 99.

manageShipping(carrier, [details])

Updates shipping company status, and account details to use when shipping.

→ carrier, details

- Carrier – must be "FedEx" or "UPS".
- Details – array, containing account number, and metric

← details

Returns the account number and metric (0 – 10, shipping preference, where 0 is highest preference).

manageAccounts(ID, [details])

Updates supplier account information.

→ ID, details

- ID – Account ID.
- details – Array, containing the name, address, city, state, zip, phone, fax, email, terms(0 – 99), contactname

← details

Returns the account details

Customer Class Functions

changePassword(ID, [pass])

Changes an account password.

→ ID, pass

- ID – Account ID.
- Pass - Password

← pass

Password hash (MD5).

search(terms)

Process search box entries.

→ terms

Terms – Array, containing each word for the search

← results

Returns item ID's matching the search result, in order of relevance, as an array.

login(user, [pass])

Login.

→ ID, pass

- ID – Account ID.
- Pass - Password

← id

Session ID.

displayProductDetails(ID)

Gets the products details from the database.

→ ID

ID – Product ID.

← details

Array with Description, Name, Price, ImageURL

addToCart(ID, [product], [count])

Product is added or removed from the shopping cart.

→ ID, product, count

- ID – session ID.
- Product – product ID number
- Count – number of items. Use "+" to increment, and "-" to decrement.

← count

If only ID supplied, an array containing the product ID's in the shopping cart is returned. If product ID is supplied, count is returned.

accountHistory(ID)

Gets past purchase IDs.

→ ID

ID – Account ID.

← details

An array containing invoice ID's of past account activity.

makePayment(ID, cardinfo, amount)

Allows charge account customers to submit payments via credit card.

→ ID, cardinfo, amount

- ID – Account ID.
- Cardinfo – Array containing credit card data
- Amount – amount of payment

← ID

Payment ID if successful, else returns FALSE.

checkOut(ID, [payment], [cardinfo])

Allows registered users to pay for items, and queue for shipping.

→ ID

ID – Session ID.

← details, amount, ID

- Details – Array containing ProductID, Price, Count.
- Amount – total amount of purchase
- ID – If payment information supplied, confirmation code returned.

shippingProfile(ID, [terms])

Allows charge account customers to change their shipping preference.

→ ID, cardinfo, amount

- ID – Account ID.
- Terms – Array containing Carrier, ServiceLevel

← terms

Returns the shipping terms as an array containing the carrier, and terms (1 – overnite, 2 – 2 next day air, 3 – ground)

paymentProfile(ID, [cardinfo])

Stores and retrieves payment profile.

→ ID, cardinfo

- ID – Account ID.
- Cardinfo – Array containing credit card data

← cardinfo

Returns the card information as an array.

order(orderID)

Gets order status, and tracking number.

→ orderID

orderID – Order ID number

← tracking

Tracking number.

shipping(ID)

Updates shipping.

→ ID

ID – Account ID.

← shipping

Orders pending shipping, as array.

createAccount(details)

Creates a new acount.

→ details

Details – an array containing lastname, firstname, userid, password, email.

← error

0 = success. 1 = failed, 2 = userID already in use.

requestInformation(name, email)

Sends an email with more information, and sends email to site admin.

→ name, email

- Name – users name
- Email – email address

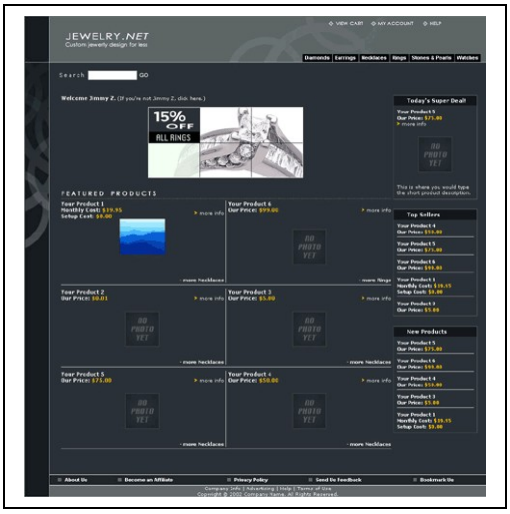
← error

0 = success, 1 = invalid email, 2 = failed.

1.6 USER INTERFACE DESIGN

1.6.1 Application

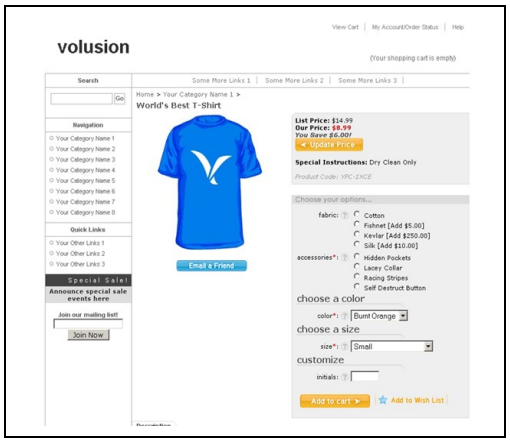
The application design concept exists of a basic e-Commerce store front, which allows visitors to browse and create accounts, as well as place orders. Below is a basic concept of the store front, as it would appear to a user on first arrival:



This is a concept only, and branding will be necessary. The customer has provided a logo for branding:

1.6.2 Browse

Browsing interface concept:



The Registered users will have access to all the customer class functions, via the web interface.

1.6.3 Management Interface

The management interface requires a secure login, and only those users with administrative attributes should have access to the admin class.

Admin Page concept: