

INTRODUCTION

The Project entitled ONLINE SHOPPING SYSTEM is a web-based application Software developed in java language using Java as front end on Pentium machine.

The main aim of ONLINE SHOPPING SYSTEM is to improve the services of Customers and vendors. It maintains the details of customer payments, product receipts, addition of new customers, products and also updating, deletion for the same. It also stores the details of invoices generated by customer and payments made by them with all Payments details like credit card.

The primary features of the project entitled ONLINE SHOPPING SYSTEM are high accuracy, design flexibility and easy availability. And also it uses database tables Representing entities and relationships between entities.

OVERVIEW OF ONLINE SHOPPING

SYSTEM

The central concept of the application is to allow the customer to shop virtually using the Internet and allow customers to buy the items and articles of their desire from the store. The information pertaining to the products are stores on an RDBMS at the server side (store). The Server process the customers and the items are shipped to the address submitted by them.

The application was designed into two modules first Os for the customers who wish to buy the articles. Second is for the storekeepers who maintains and updates the information pertaining to the articles and those of the customers?

The end user of this product is a departmental store where the application is hosted on the web and the administrator maintains the database. The application which is deployed at the customer database, the details of the items are brought forward from the database for the customer view based on the selection through the menu and the database of all the products are updated at the end of each transaction.

Data entry into the application can be done through various screens designed for various levels of users. Once the authorized personnel feed the relevant data into the system, several reports could be generated as per the security.

PROBLEM DEFINITION

To develop a web-based application to improve the service to the customers and merchant which in turn increases the sales and profit in "ONLINE SHOPPING"

GOALS FOR THE SYSTEM AND THE PROJECT:

The system is capable of maintaining details of various customers, vendors, Products and storing all the day to day transactions such as generation of shipment address bills, handling customers and product receipts, updating of stores.

CONSTRAINTS ON THE SYSTEM AND THE PROJECT :

ONLINE SHOPPING is developed in Java 1.2.2 using Java as front end and it could run only on Java 1.2 and onward versions.

HARDWARE AND SOFTWARE REQUIREMENTS

1. Hardware Requirements

- PENTIUM SERVER with Network Dx4 300Mhz
- 8.3 GB of hard disk memory space
- 512 MB of RAM
- Monitor
- Keyboard
- Mouse

2. Software Requirements

- Operating System : Windows xp or window NT
- Editor : Visual Studio 2008 or higher

- Browser : Internet Explorer, Mozilla Firefox, any
- Front End : Asp.net, HTML, Java Script
- Back End : SQL server 2005

CONCEPTS AND TECHNIQUES

The Java Packages

Eight packages comprise the standard Java development environment.

The Java Language Package

The Java language package, also known as java.lang, contains classes that are core to the Java language. The classes in this package are grouped as follows:

Object

The granddaddy of all classes--the class from which all others inherit.

Data Type Wrappers

A collection of classes used to wrap variables of a primitive data type: Boolean, Character, and Double, Float Integer and Long.

Strings

Two classes that implement character data. The String and String Buffer Classes is a thorough lesson on the use of both types of strings.

System and Runtime

These two classes provide let your programs use system resources. System provides a system-independent programming interface to system resources and Runtime gives you direct system-specific access to the runtime environment. Using System Resources Describes both the System and Runtime classes and their methods.

Process

Process objects represent the system process that is created when you use Runtime to execute system commands. The java.lang Packages defines and implements the generic Process class.

The compiler automatically imports this package for you. No other packages are automatically imported.

The Java I/O Package

The Java I/O Package (java.io) provides a set of input and Output streams used to read and write data to files or other Input and output sources. The classes and interfaces defined In java.io are covered fully in Input and Output Streams.

The Java Utility Package

This Java package, java.util, contains a collection of utility classes. Among them are several generic data structures (Dictionary, Stack, Vector, and Hash table) a useful object for tokenizing a string and another for manipulating calendar dates.

The java.util package also contains the Observer interface and Observable class, which allow objects to notify one another when they change. The java.util classes aren't covered separately in this tutorial although some examples use these classes.

The Java Networking Package

The java.net package contains classes and interface definitions that implement various networking capabilities. The

Classes in this package include a class that implement a URL, a connection to a URL, a socket connection, and a datagram packet. You can use these classes to implement client-server applications and other networking communication applications.

Custom Networking and Security has several examples using these classes, including a client-server example and an example that uses datagrams.

The Applet Package

This package contains the Applet class -- the class that you must subclass if you're writing an applet. Included in this

Package is the Audio Clip interface which provides a very high level abstraction of audio. Writing Applets explains the ins and outs of developing your own applets.

INTRODUCTION TO JDBC

What is JDBC and Why JDBC?

JDBC (Java Database Connectivity) is a front-tool for connecting to a server and is similar to ODBC in the respect. However, JDBC can connect only Java clients and uses ODBC for the connectivity. JDBC is essentially a low-level Application, Programming Interface. It is called a low-level API since any data manipulation, storage and retrieval has to be done by the program itself. Some tools that provide a higher level abstraction are expected shortly.

JDBC Driver types: -

There are four types of JDBC drivers each having its own functionality. Please note that, they do not substitute one another, each having their own suitability aspects. They are classified based on how they access data from the database.

1. Native Jdbc driver:

A JDBC driver, which is partly written in Java and most of each, implemented using native methods to access the database. This is useful in case of Java application that can run only on some specific platforms. Writing these type drivers is easier compare to writing other drivers

2. All Java JDBC Net Drivers:

A JDBC net drivers which uses a common network protocol to connect to an

intermediate server, which is turn employees native calls to connect to the data base.

This approach is used for applets where the request must go through the intermediate server.

3. JDBC-ODBC bridge driver:

A bridge driver provided with JDBC can convert. The Jdbc calls in to equaling ODBC calls using the native methods. Since ODBC provides for connection to any type of database that is ODBC compliant, to connect a number of databases simultaneously, it is very simple matter. This approach is a recommended once since using ODBC drivers, which are industry standard as of now, would make an application truly portable across the databases.

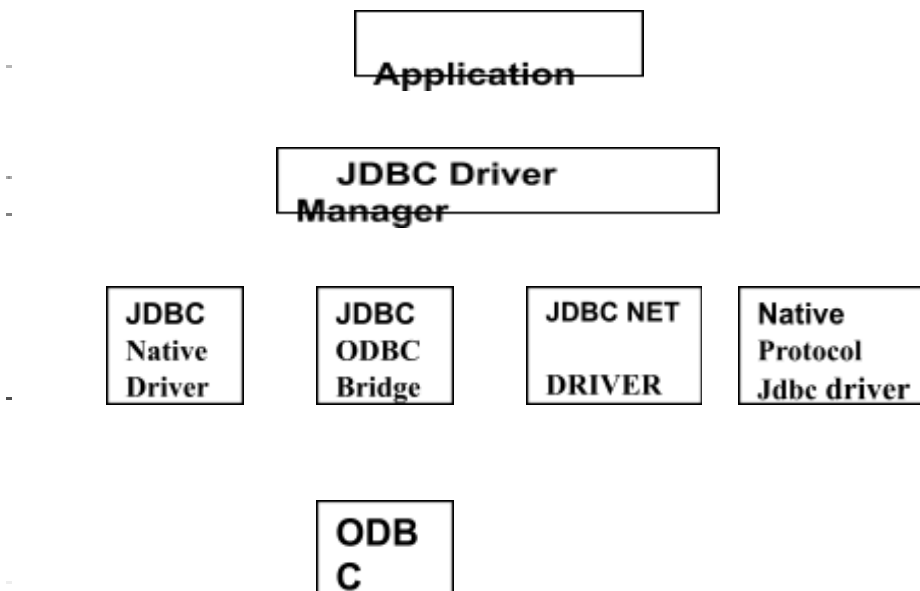
4. Native Protocol ALL Java Drivers: -

This type of Jdbc driver is written completely in Java and can access the database by making use of native protocols of the database. This method of data access is suitable in case of Intranets where carry every thing can run as an application instead of an applet.

JDBC ARCHITECTURE:

JDBC architecture is shown below. Components of JDBC are

1. JDBC Driver Manager
2. JDBC Driver
3. JDB-ODBC Bridge
4. Applications.



DATA BASE

JDBC Driver Manager: -

Function of the driver manager is to findout available drivers in the system and connects the application to the appropriate database. Whenever a connection is requested. However, to help the driver manager identify different types of drivers, each driver should be registered with the driver Manager.

JDBC Driver: - Function of the JDBC Driver is to accept the SQL calls from the application and convert them into native calls to the database. However, in this process it may take help some other drivers or even servers, which depends on the type of Jdbc Driver we are using. Also, it is possible that the total functionality of the database server could be built into the driver itself.

JDBC-ODBC Bridge: -

Sunsoft provides a special JDBC Driver called JDBC-ODBC bridge driver, which can be used to connect to any existing database, that is ODBC complaint.

APPLICATION: Application is a Java Program that needs the information to be modified in some database or wants to retrieve the information.

SERVLETS:

Client and Servers

To understand the World Wide Web and Server Side programming must understand the division between Web clients and Web servers and how HTTP facilities the interaction between the two. Putting in simple words, a server handles request from various clients. For example, suppose you are using a word processing program to edit files on another compute. Your computer would be the client because it is requesting the files from another computer. The other computer would be the server because it is handling your computer's request. With networked computers, clients and servers are very common. A server typically runs on a

different machine than the client, although this not always the case. The interaction between the two usually begins on the client side. The client software request an object or transaction from the server software, which either handles the request or denies it. If the request is handled, the object is sent back to the client software. On the World Wide Web server are known as web servers, and clients are known as Web browsers. Web browser's request documents from web servers, allowing you to view documents on the World Wide Web. Some of the most common browsers are Netscape Navigator, Internet Explorer, and NCA's Mosaic. Like most software companies that distribute Web Browsers, these companies also distribute Web server software (in our case we uses JWS Java Web Browser). The process of viewing a document on the web starts when a web browser sends a request to web server in http request headers. The Web server receives and views the http request headers for any relevant information, such as the name of the file being requested, and sends back the file with HTTP response headers. The web browser then uses the HTTP response headers to determine how to display the file or data being returned by the web server.

In the case of Servlets we would refer to a URL with a default port at 8080. If the above said server are Java Enabled Web Servers. HTTP is defaulted at port 80 and in case of Servlets it is defaulted at 8080 that could be changed.

WHY USE SERVLETS?

You may have noticed that the preceding CGI example could just as easily been a simple HTML file. In fact, you don't gain advantage from making it a server side script. So why use any server side scripting? Well, for the preceding example, you wouldn't. It is just a simple example CGI script. As you learn Servlets, however, you will see that it allows you to extend the functionality of Web documents to produce dynamic and interactive pages.

1.An Invitation to Servlets:

This session provides answer to the questions "What is Servlets" shows typical uses for Servlets. It also gives a quick introduction to HTTP and its implementation in the HttpServlet class.

What is Servlets?

Servlets are modules of Java code that run in a server application to answer client requests. Servlets are not tied to a specific client-server protocol but they are most commonly used with Http and the word "SERVLET" is often used in the meaning of "HTTP Servlets".

Servlets make use of the Java standard extension classes in the packages javax.servlet and javax.servlet.http (extension of the Servlets framework for Servlets that answer HTTP requests). Since Servlets are written in the highly

portable Java Language and follow a standard framework, they provide a means to create server extensions in a server and operating system independent way.

Typical uses for HTTP Servlets include:

- Processing and/or storing data submitted an HTML form.
- Providing synergic content, e.g. returning the results of a database query to the client.
- Managing state information on top of the stateless HTTP, e.g. for an online shopping chart system which manages shopping charts for many concurrent customers and maps every request to the right customer.

The Servlet Environment:

Inter-Servlet communication

The inter Servlet communication method which is described in this section can only be used with Servlet engines which implement version 1.0 or 2.0 of the Servlet API. It will not work with Servlet engines, which comply strictly, to version 2.1

Servlets are not alone in a web server. They have access to other Servlets in the same Servlet context (usually a Servlet directory), represented by an instance of `javax.servlet.servletContext`. The Servlet Context is available through the Servlet Config object's `get Servlet context` method.

A Servlet can get a list of all othg3r Servlets in the Servlet context by calling `get Servlet Names` on the Servlet Context object. A Servlet for a known name (probably obtained through `getServletNames`) is returned by `getServlet`. Note that this method can throw a `ServletException` because it may need to load and initialize the requested `ServletException` if this was not already done.

After obtaining the reference to another Servlet that Servlets methods can be called. Methods, which are not declared in `javax.servlet.Servelet` but in a subclass thereof can, called by casting the returned object to the required class type.

Note that in Java the identity of a class is not only defined by the class name but also by the Class Loader by which it was loaded. Web servers usually load each Servlet with a different class loader. This is necessary to reload Servlets on the fly because single classes cannot be replaced in the running JVM. Only a classloader object with all loaded classes can be replaced.

This means that classes, which are loaded by a Servlet class loader, cannot be used for inter-Servlet communication. A class literal `FooServlet` (as used in a type cast like `"FooServlet foo ((FooServlet) context.getServlet ("FooServlet"))"`) which is used in class `BarServlet` is different from the class literal `FooServlet` as used in `FooServlet` itself.

A way to overcome 5this problem is using a supercalss or and interface which is loaded by the system loader and thus shared by all Servlets. In a Web Server, which is written in Java, these classes are usually located in the class path (as

defined by the CLASSPATH environment variable)

Introduction to world wide web

Internet:

The Internet is a network. It consists of thousands of interconnected networks consisting of different types of computers. It is the worldwide access to people and information.

The internet stemmed from the concept of universal database: data that would not only be accessible to people around the world, but information that would link easily to others pieces of information so that the relevant data can be quickly found by a user.

In 1960s this was explored revolution all aspects of human-information interaction. The Internet first began in 1969 when the United States Department of Defense researched ways of communication via decentralized computer networks. After that computers in government and universities were voluntarily linked. This quickly spread to other institutions around the world where today there are an estimated 20 million computers linked together.

UNDERSTANDING INTERNET

A network in which computers are connected using cables or some other direct media are said to be in LAN. When a group of LAN's are connected together they are called Wan's. These Wan's are connected via telephone line, satellite links etc. Importantly these LAN's are connected to form Wan's through special computers called Routers.

The job of a Router is to provide a link from one network to other where networks can be LAN's to form Wan's to become even larger Wan's.

HARDWARE REQUIREMENTS:

If Internet being accessed through a telephone line
Then we need

- A computer with minimum requirements

- A modem
- A telephone

Taking a first issue of computers with minimum requirements may put everyone wondering what could be the requirements?
It would be safe enough to use the 486 or higher processors for a computer to be fast enough with at least 16MB to 512MB MEMORY.

SYSTEM SPECIFICATION

The application ONLINE SHOPPING was designed into two modules

1. For the customers who wish to buy the articles?
2. For the storekeeper who maintains and updates the information pertaining to the articles and those of the customers.

BUSSINESS PROCESS MODEL

The end-user of this product is a departmental store where the application is hosted on the web and the administrator maintains the database. The application which is deployed at the departmental stores will automate the customer details that are appended to the customers database, the details of the items are brought forward from the database for the customers view based on the selection through the menu and the database of all the products are updated at the end of each transaction.

Data entry into the application can be done through various screens designed for various levels of users. Once the authorized personnel feed relevant data into the system, several reports could be generated as per the security.

If the customer wants to buy the product he wants to enter into the shopping chart. He selects the desired product, after that he enter in to the order form he fills the order form. After filling order form merchant shipped the product to address specified by the Customer.

Bill is send by merchant the address specified by the customer, after that customer gives the credit card number. Here marchant can be stored the customer shipped address, and customer details.

Initially marchant enter with login, which is maintained in database, and also maintain the product details in the database. Every time he has access the

insertion Updating and deletion when and where ever he wants.

The central concept of the application is to allow the Customer to shop virtually using the Internet and allow customers to buy the items and articles of their desire form the store. The information pertaining to the products are stored on an RDBMS at the server side (store). The Server processes the customer's request and the items are shipped to the address submitted by them.

HTML is a formal set of specifications used to define information, which can be added to the content of a document as an aid processing. The purpose of HTML is to specify how the text should be processed. The purpose of a browser becomes to act as a presentation engine, to interpret HTML & display the contents in appropriate manner.

Within Hyper text there will be an indication of the start and end of the paragraph, there will be no explicit instructions about the size, fonts, color, etc., It will contain the basic information about the link but no specifications about how to display the highlight the link. It left to the browser how information should be displayed.

The importance of HTML lies with Internet because one doesn't know about the end user since there are different user types on the NET. If HTML would be a language having instructions about the colors, sizes & fonts it would be subjected to a specific environment> Secondly as the number of users grow day by day one should think of minimizing the amount of information to be passed from one system to the other. Hence HTML would contain plan ext. file would occupy less space than any other graphics file but, having the addresses of those locations.

If hypertext one must send information for displaying the text and these instructions are embedded in the text itself.

SYSTEM DESIGN

Design of software involves conceiving, planning out and specifying the externally observable characteristics of the software product. We have data design, architectural design and user interface design in the design process. These are explained in the following section. The goal of design process is to provide a blue print for implementation, testing and maintenance activities.

DATA DESIGN:

The primary activity during data design is to select logical representations of data objects identified during requirement analysis and software analysis. A data dictionary explicitly represents the relationships among data objects and the constraints on the elements of the data structure. A data dictionary should be established and used to define both data and program design.

FEASIBILITY STUDY:

Feasibility study is conducted once the problem is clearly understood. Feasibility study is a high level capsule version of the entire system analysis and design process. The objective is to determine quickly at a minimum expense how to solve a problem. The purpose of feasibility is not to solve the problem but to determine if the problem is worth solving. The system has been tested for feasibility in the following points.

1. Technical Feasibility
2. Economical Feasibility
3. Operational Feasibility.

1. Technical Feasibility: -

The project entitles "Project Monitoring System" is technically feasibility because of the below mentioned feature. The project was developed in Java which Graphical User Interface.

It provides the high level of reliability, availability and compatibility. All these make Java an appropriate language for this project. Thus the existing software Java is a powerful language.

2. Economical Feasibility: -

The computerized system will help in automate the selection leading the profits and details of the organization. With this software, the machine and manpower utilization are expected to go up by 80-90% approximately. The costs incurred of not creating the system are set to be great, because precious time can be wanted by manually.

3. Operational Feasibility:

In this project, the management will know the details of each project

where he may be presented and the data will be maintained as decentralized and if any inquires for that particular contract can be known as per their requirements and necessities.

Implementation:

Implementation is the stage where the theoretical design is turned into a working system. The most crucial stage in achieving a new successful system and in giving confidence on the new system for the users that it will work efficiently and effectively.

The system can be implemented only after thorough testing is done and if it is found to work according to the specification.

It involves careful planning, investigation of the current system and its constraints on implementation, design of methods to achieve the change over and an evaluation of change over methods a part from planning. Two major tasks of preparing the implementation are education and training of the users and testing of the system.

The more complex the system being implemented, the more involved will be the systems analysis and design effort required just for implementation.

The implementation phase comprises of several activities. The required hardware and software acquisition is carried out. The system may require some software to be developed. For this, programs are written and tested. The user then changes over to his new fully tested system and the old system is discontinued.

TESTING:

The testing phase is an important part of software development. It is the process of finding errors and missing operations and also a complete verification to determine whether the objectives are met and the user requirements are satisfied.

Software testing is carried out in three steps:

The first includes unit testing, where in each module is tested to provide its correctness, validity and also determine any missing operations and to verify whether the objectives have been met. Errors are noted down and corrected immediately. Unit testing is the important and major part of the project. So errors are rectified easily in particular module and program clarity is increased. In this

project entire system is divided into several modules and is developed individually. So unit testing is conducted to individual modules.

The second step includes Integration testing. It need not be the case, the software whose modules when run individually and showing perfect results, will also show perfect results when run as a whole. The individual modules are clipped under this major module and tested again and verified the results. This is due to poor interfacing, which may results in data being lost across an interface. A module can have inadvertent, adverse effect on any other or on the global data structures, causing serious problems.

The final step involves validation and testing which determines which the software functions as the user expected. Here also some modifications were. In the completion of the project it is satisfied fully by the end user.

Maintenance and Enhancement

AS the number of computer based systems, grievance libraries of computer software began to expand. In house developed projects produced tones of thousand soft program source statements. Software products purchased from the outside added hundreds of thousands of new statements. A dark cloud appeared on the horizon. All of these programs, all of those source statements-had to be corrected when false were detected, modified as user requirements changed, or adapted to new hardware that was purchased. These activities were collectively called software Maintenance.

The maintenance phase focuses on change that is associated with error correction, adaptations required as the software's environment evolves, and changes due to enhancements brought about by changing customer requirements. Four types of changes are encountered during the maintenance phase.

Correction

Adaptation

Enhancement

Prevention

Correction:

Even with the best quality assurance activities is lightly that the customer will uncover defects in the software. Corrective maintenance changes the software to correct defects.

Maintenance is a set of software Engineering activities that occur after software has been delivered to the customer and put into operation. Software configuration management is a set of tracking and control activities that began

when a software project begins and terminates only when the software is taken out of the operation.

We may define maintenance by describing four activities that are undertaken after a program is released for use:

Corrective Maintenance

Adaptive Maintenance

Perfective Maintenance or Enhancement

Preventive maintenance or reengineering

Only about 20 percent of all maintenance work are spent "fixing mistakes". The remaining 80 percent are spent adapting existing systems to changes in their external environment, making enhancements requested by users, and reengineering an application for use.

ADAPTATION:

Over time, the original environment (E>G., CPU, operating system, business rules, external product characteristics) for which the software was developed is likely to change. *Adaptive maintenance* results in modification to the software to accommodate change to its external environment.

ENHANCEMENT:

As software is used, the customer/user will recognize additional functions that will provide benefit. Perceptive maintenance extends the software beyond its original function requirements.

PREVENTION:

Computer software deteriorates due to change, and because of this, preventive maintenance, often called software re engineering, must be conducted to enable the software to serve the needs of its end users. In essence, preventive maintenance makes changes to computer programs so that they can be more easily corrected, adapted, and enhanced. Software configuration management (SCM) is an umbrella activity that is applied throughout the software process. SCM activities are developed to

Identify change.

Control change.

Ensure that change is being properly implemented.

Report change to others that may have an interest.

DESIGN SPECIFICATION

TABLES

Table Name: PRODUCT

Column Name	Type
Product Id	Int
Product Name	Varchar
Product Type	Varchar
Product Price	Float
Quantity	Varchar
Units in stock	Int
Units on Order	Int

Table Name: CUSTOMER

Column Name	Type
ID	Int
Order Number	Int
Order Date	Date
CustomerName	Varchar
BillAddress	Varchar
ShippingAddress	Varchar
PhoneNumber	Varchar
Email-Id	Varchar

Amount	Float
Shipped Status	Varchar

Table Name: PRODTABLE

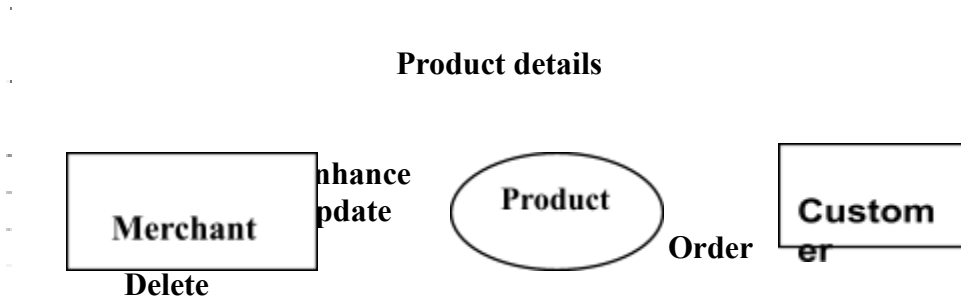
Column Name	Type
ProdId	Int
ProductName	Varchar
Supplier	Varchar
Price	Float

Table Name: CREDITCARD

Column Name	Type
CardType	Varchar
CardNumber	Varchar

DATAFLOW DIAGRAMS

CONTEXT DIAGRAM



Customer Details

FIRST LEVEL DATA FLOW DIAGRAM FOR MERCHANT LOGIN TO ENHANCE STORES



Store Details

FIRST LEVEL DATA FLOW DIAGRAM FOR MERCHANT LOGIN TO ENHANCE STORES

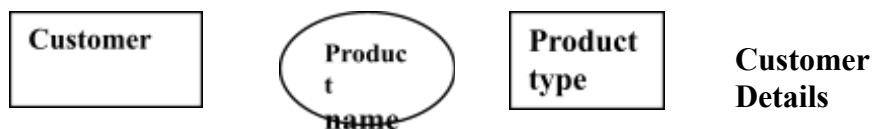
Merchant Details



Updated store
Details

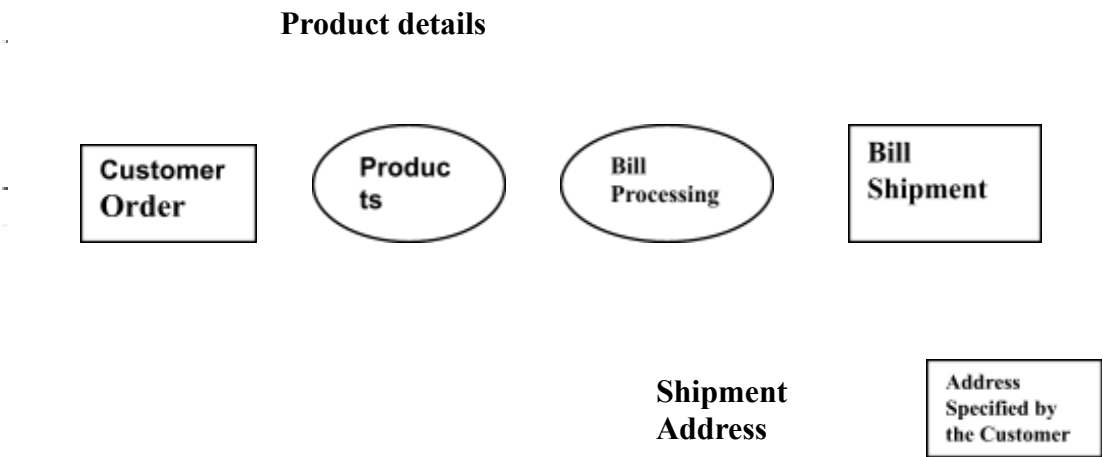
FIRST LEVEL DATAFLOW DIAGRAM FOR CUSTOMER PROCESSING ON PRODUCTS

Product details

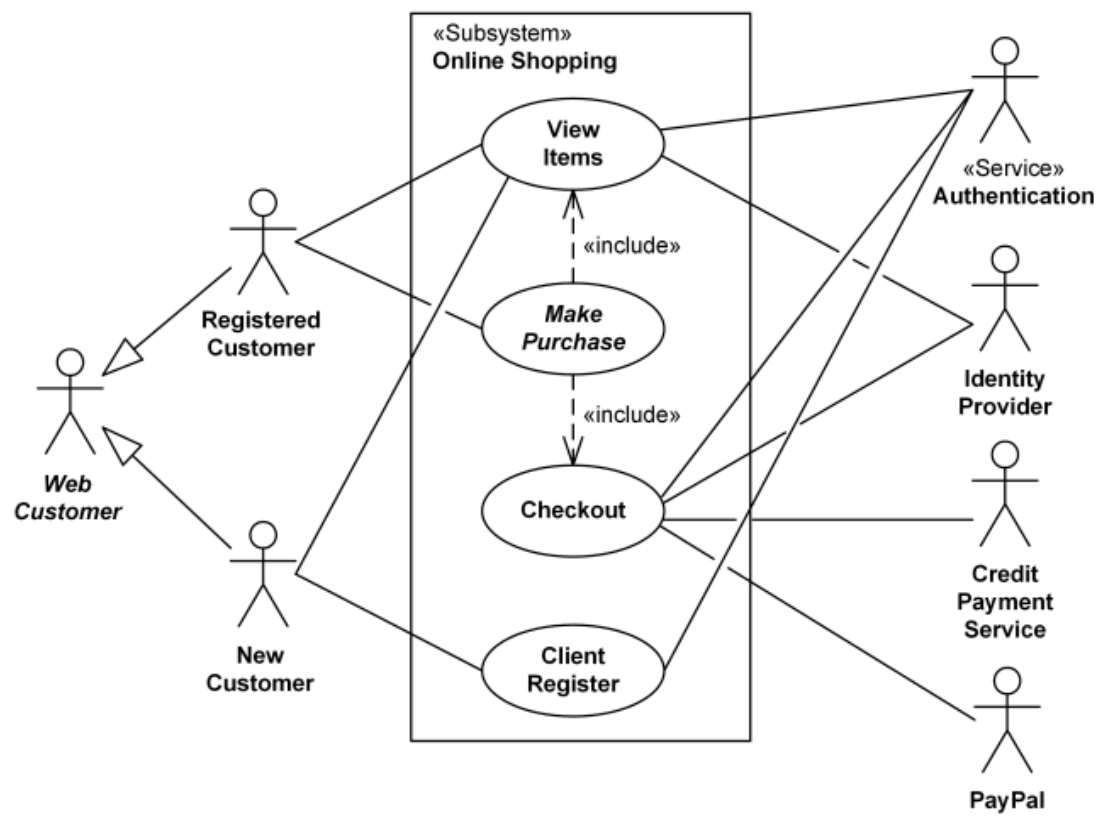


Amount Payable

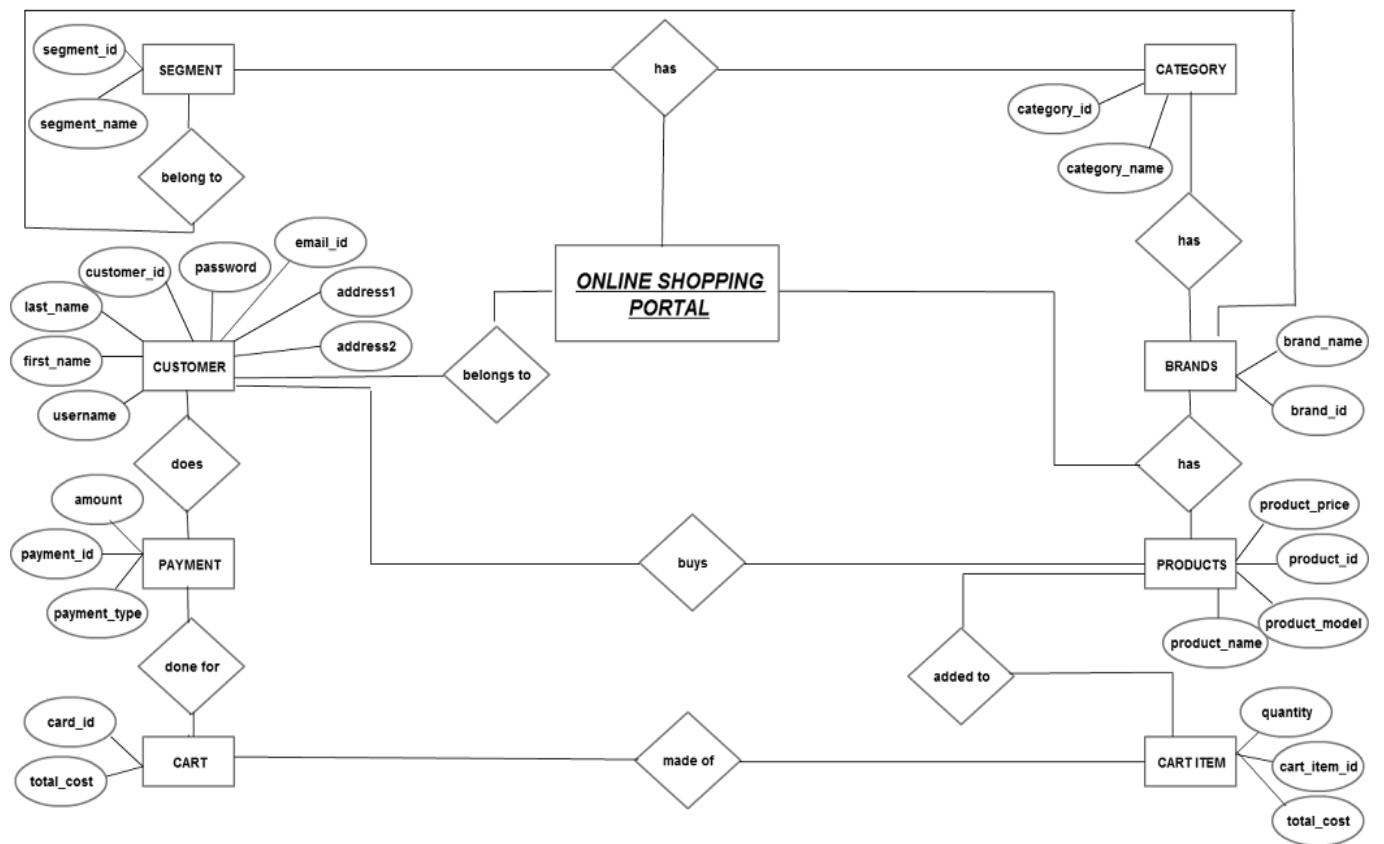
SECOND LEVEL DATA FLOW DIAGRAM FOR BILL SHIPMENT



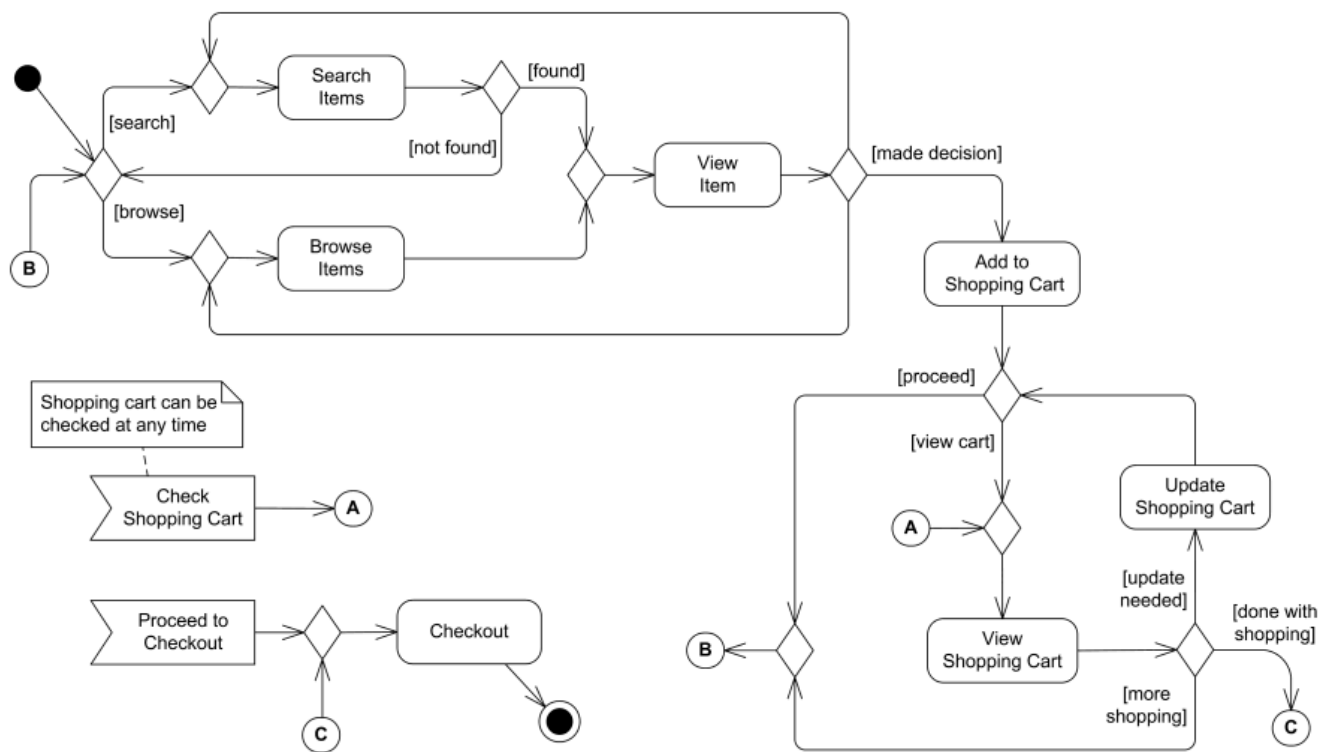
USE Case Diagram:



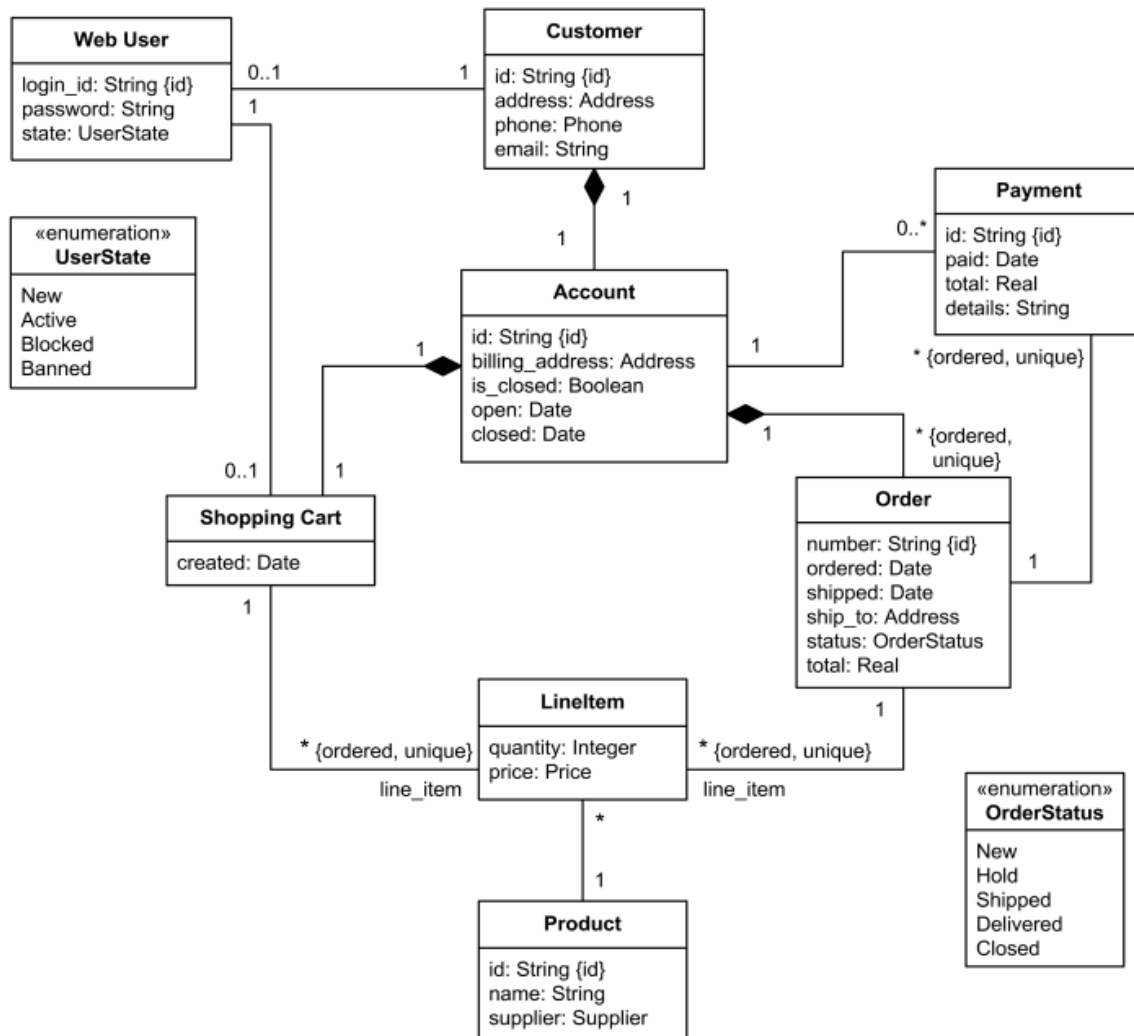
ER Diagram:



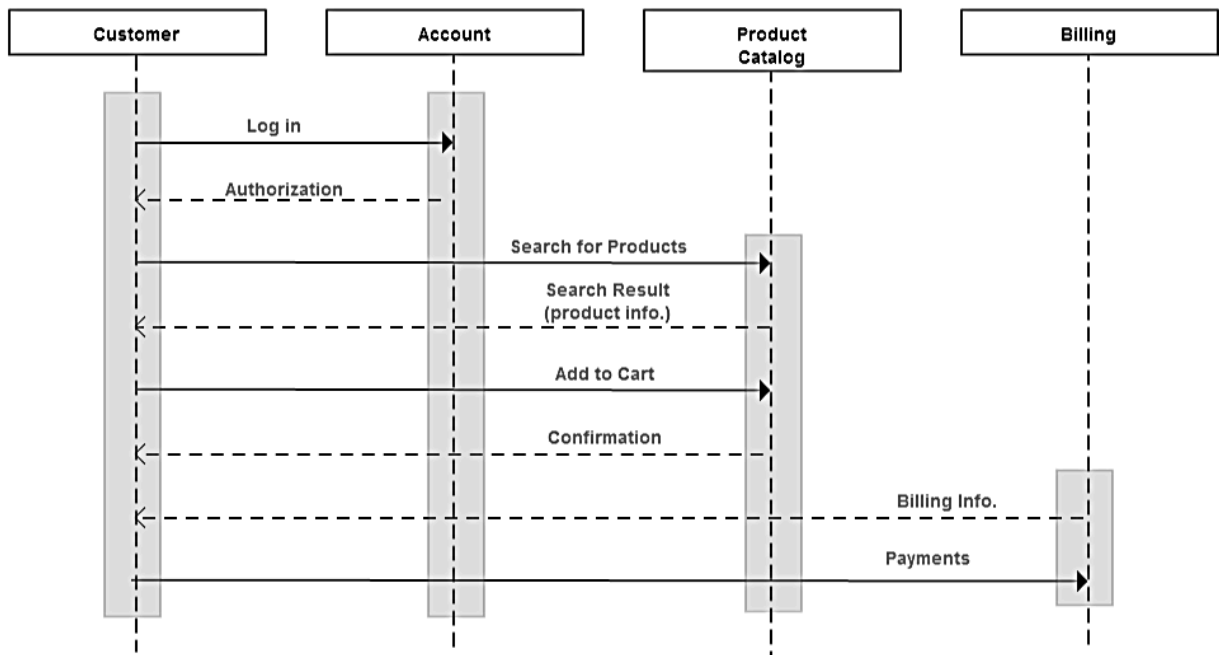
Activity Diagram:



Class Diagram:



Sequence Diagram:



CODDING

The layout

We won't spend much time to create a nice layout for our sample shop. Just a base class for the controllers and a few lines of CSS code will be enough for now:

```
//controller/shopbase.class.php
<?php
use ScavixWDF\Base\HtmlPage;

class ShopBase extends HtmlPage { /* no code needed here */ }
?>
// controller/shopbase.tpl.php
<div id="page">
    <div id="navigation">
        <a href="<?=buildQuery('Products')?>">Products</a>
        <a href="<?=buildQuery('Basket')?>">Basket</a>
        <a href="<?=buildQuery('Admin')?>">Administration (normally hidden)</a>
    </div>
    <div id="content">
        <? foreach( $content as $c ) echo $c; ?>

// res/shopbase.css
#page { font: 14px normal Verdana,Arial,serif; }
#page > div { width: 960px; margin: auto; }
#navigation { padding-bottom: 10px; border-bottom: 1px solid gray; }
#navigation a
{
    font-size: 18px;
    font-weight: bold;
    margin-right: 15px;
}

.product_overview
{
    clear: both;
```

```

        margin-top: 25px;
        border-bottom: 1px solid gray;
        height: 75px;
    }

    .product_overview img
    {
        width: 50px;
        float: left;
        margin-right: 10px;
    }

    .product_overview div
    {
        white-space: nowrap;
        overflow: hidden;
    }

    .product_overview .title { font-weight: bold; }
    .product_overview a { float: right; }

    .product_basket
    {
        clear: both;
    }

    .product_basket img
    {
        width: 30px;
        float: left;
        margin-right: 10px;
    }

    .basket_total
    {
        clear: both;
        text-align: right;
        font-size: 14px;
        font-weight: bold;
    }

```

Products

We will implement two pages here: A product listing and a product details page

```

// controller/products.class.php
<?php
use ScavixWDF\Base\Template;
use ScavixWDF\JQueryUI\uiMessage;

class Products extends ShopBase

```

```

{
    /**
     * Lists all products.
     * @attribute[RequestParam('error','string',false)]
     */
    function Index($error)
    {
        // display error message if given
        if( $error )
            $this->content(uiMessage::Error($error));

        // loop thru the products...
        $ds = model_datasource('system');
        foreach( $ds->Query('products')->orderBy('title') as $prod )
        {
            //... and use a template to represent each
            $this->content( Template::Make('product_overview') )
                ->set('title',$prod->title)
                ->set('tagline',$prod->tagline)
            // see config.php where we set up products images folder as resource folder
            ->set('image',resFile($prod->image))
            ->set('link',buildQuery('Products','Details',array('id'=>$prod->id)))
            ;
        }
    }

    /**
     * Shows product details
     * @attribute[RequestParam('id','int')]
     */
    function Details($id)
    {
        // check if product really exists
        $ds = model_datasource('system');
        $prod = $ds->Query('products')->eq('id',$id)->current();
        if( !$prod )
            redirect('Products','Index',array('error'=>'Product not found'));

        // create a template with product details
        $this->content( Template::Make('product_details') )
            ->set('title',$prod->title)
            ->set('description',$prod->body)
            // see config.php where we set up products images folder as resource folder
            ->set('image',resFile($prod->image))
            ->set('link',buildQuery('Basket','Add',array('id'=>$prod->id)))
            ;
    }
}

// templates/product_overview.tpl.php
<div class="product_overview">

```

```

<div class="title"><?=$title?>
```

The 'Details' method uses a similar approach but does not loop, but load a single product. It also uses another template file.

```
// templates/product_details.tpl.php
<div class="product_details">
  
  <div class="title"><?=$title?>
```

The products listing page:

```
// controller/basket.class.php
<?php
use ScavixWDF\Base\Template;
use ScavixWDF\jQueryUI\uiButton;
use ScavixWDF\jQueryUI\uiMessage;

class Basket extends ShopBase
{
    /**
     * Lists all items in the basket.
     * @attribute[RequestParam('error','string',false)]
     */
    function Index($error)
    {
        // display any given error message
        if( $error )
            $this->content(uiMessage::Error($error));

        // prepare basket variable
        if( !isset($_SESSION['basket']) )
            $_SESSION['basket'] = array();

        if( count($_SESSION['basket']) == 0 )
            $this->content(uiMessage::Hint('Basket is empty'));
        else
        {
            // list all items in the basket ...
            $sds = model_datasource('system');
            $price_total = 0;
            foreach( $_SESSION['basket'] as $id=>$amount )
            {
                $prod = $sds->Query('products')->eq('id',$id)->current();

                //... each using a template
                $this->content( Template::Make('product_basket') )
                    ->set('title',$prod->title)
                    ->set('amount',$amount)
                    ->set('price',$prod->price)
                // see config.php where we set up
```



```

        // products images folder as resource folder
        ->set('image',resFile($prod->image))
        ->set('add',buildQuery('Basket','Add',array('id'=>$prod->id)))
        ->set('remove',buildQuery('Basket','Remove',array('id'=>$prod->id)))
        ;
        $price_total += $amount * $prod->price;
    }
    // display total price and the button to go on
    $this->content("<div
    class='basket_total'>Total price: $price_total</div>");
    $this->content( uiButton::Make("Buy now") )->onclick =
    "location.href = '".buildQuery('Basket','BuyNow')."'";
    }
}

/**
 * Adds a product to the basket.
 * @attribute[RequestParam('id','int')]
 */
function Add($id)
{
    // check if the product exists
    $ds = model_datasource('system');
    $prod = $ds->Query('products')->eq('id',$id)->current();
    if( !$prod )
        redirect('Basket','Index',array('error'=>'Product not found'));

    // increase the counter for this product
    if( !isset($_SESSION['basket'][$id]) )
        $_SESSION['basket'][$id] = 0;
    $_SESSION['basket'][$id]++;
    redirect('Basket','Index');
}

/**
 * Removes an item from the basket.
 * @attribute[RequestParam('id','int')]
 */
function Remove($id)
{
    // check if the product exists
    $ds = model_datasource('system');
    $prod = $ds->Query('products')->eq('id',$id)->current();
    if( !$prod )
        redirect('Basket','Index',array('error'=>'Product not found'));

    // decrease the counter for this product
    if( isset($_SESSION['basket'][$id]) )
        $_SESSION['basket'][$id]--;
    // and unset if no more items left
    if( $_SESSION['basket'][$id] == 0 )

```

```

        unset($_SESSION['basket'][$id]);
        redirect('Basket','Index');
    }

```

/* full code:

```

https://github.com/ScavixSoftware/WebFramework/blob/master/web/sample\_shop/controller/basket.class.php */
}

```

Finally to mention: Again we use a special template for the order items' listing:

```

// templates/product_basket.tpl.php
<div class="product_basket">
    
    <span class="title"><?=$title?>
// controller/basket.class.php
<?php
use ScavixWDF\Base\Template;
use ScavixWDF\jQueryUI\uiButton;
use ScavixWDF\jQueryUI\uiMessage;

```

```

class Basket extends ShopBase
{

```

/* full code:

```

https://github.com/ScavixSoftware/WebFramework/blob/master/web/sample\_shop/controller/basket.class.php */

```

```

/**
 * Entrypoint for the checkout process.
 *
 * Requests customers address details and asks for payment processor.
 */

```

```

function BuyNow()
{
    // displays the checkout form, which has all inputs for address on it
    $this->content( Template::Make('checkout_form') );
}

```

```

/**
 * Persists current basket to the database and starts checkout process.
 * @attribute[RequestParam('fname','string')]
 * @attribute[RequestParam('lname','string')]
 * @attribute[RequestParam('street','string')]
 * @attribute[RequestParam('zip','string')]
 * @attribute[RequestParam('city','string')]
 * @attribute[RequestParam('email','string')]
 * @attribute[RequestParam('provider','string')]
 */

```

```

function StartCheckout($fname,$lname,$street,$zip,$city,$email,$provider)
{
    if( !$fname || !$lname || !$street || !$zip || !$city || !$email )
        redirect('Basket','Index',array('error'=>'Missing some data'));
}

```

```

// create a new customer. note that we do not check for existence or stuff.
// this should be part of a real shop system!
$cust = new SampleCustomer();
$cust->fname = $fname;
$cust->lname = $lname;
$cust->street = $street;
$cust->zip = $zip;
$cust->city = $city;
$cust->email = $email;
$cust->price_total = 0;
$cust->Save();

// create a new order and assign the customer (from above)
$order = new SampleShopOrder();
$order->customer_id = $cust->id;
$order->created = 'now()';
$order->Save();

// now loop thru the basket-items and add them to the order...
$dss = model_datasource('system');
foreach( $_SESSION['basket'] as $id=>$amount )
{
    //... by creating a dataset for each item
    $prod = $dss->Query('products')->eq('id',$id)->current();
    $item = new SampleShopOrderItem();
    $item->order_id = $order->id;
    $item->price = $prod->price;
    $item->amount = $amount;
    $item->title = $prod->title;
    $item->tagline = $prod->tagline;
    $item->body = $prod->body;
    $item->Save();

    $order->price_total += $amount * $prod->price;
}
// save the order again to persist the total amount
$order->Save();
$_SESSION['basket'] = array();

// finally start the checkout process using the given payment provider
log_debug("Handing control over to payment provider '$provider'");
$sp = new $provider();
$sp->StartCheckout($order,buildQuery('Basket','PostPayment'));
}

/**
 * This is the return URL for the payment provider.
 * Will be called when payment reaches a final state, so control is handed over to our
 * app again from the payment processor.
 */

```

```

function PostPayment()
{
    // we just display the $_REQUEST data for now.
    // in fact this is the point where some processing
    // should take place: send email to the team,
    // that prepares the items for shipping, send email(s) to customer,...
    log_debug("PostPayment",$_REQUEST);
    $this->content("<h1>Payment processed</h1>");
    $this->content("Provider returned this data:<br/>" +
        "<pre>".render_var($_REQUEST)."</pre>");
}

/**
 * This is a special handler method for PayPal.
 * It will be called asynchronously from PayPal
 * backend so user will never see results of it.
 * Just here to update the database when payments
 * are ready or refunded or whatever.
 * See https://www.paypal.com/ipn for details
 * but in fact WebFramework will handle this for you.
 * Just needs this entry point for the callback.
 * @attribute[RequestParam('provider','string')]
 */
function Notification($provider)
{
    log_debug("Notification",$_REQUEST);
    $provider = new $provider();
    if( $provider->HandleIPN($_REQUEST) )
        die("OK");
    die("ERR");
}
}

// model/sampleshoporder.class.php
<?php
use ScavixWDF\Model\Model;
use ScavixWDF\Payment\IShopOrder;
use ScavixWDF\Payment\ShopOrderAddress;

/**
 * Represents an order in the database.
 *
 * In fact nothing more than implementations for the inherited Model
 * and the implemented IShopOrder interface.
 * See
https://github.com/ScavixSoftware/WebFramework/wiki/classes\_modules\_payment#wiki-1c67f96d00c3c22f1ab9002cd0e3acbb
 * More logic would go into the Set* methods to handle different order states.
 * For our sample we just set the states in the DB.
 */
class SampleShopOrder extends Model implements IShopOrder
{

```

```

const UNKNOWN = 0;
const PENDING = 10;
const PAID = 20;
const FAILED = 30;
const REFUNDED = 40;

```

```

/**
 * Returns the table name.
 * See

```

https://github.com/ScavixSoftware/WebFramework/wiki/classes_essentials_model_model.class#gettablename

```

 */
public function GetTableName() { return 'orders'; }

```

```

/**
 * Gets the orders address.
 * @return ShopOrderAddress The order address
 */

```

```

public function GetAddress()
{
    $res = new ShopOrderAddress();
    $res->Firstname = $this->fname;
    $res->Lastname = $this->lname;
    $res->Address1 = $this->street;
    $res->Zip = $this->zip;
    $res->City = $this->city;
    $res->Email = $this->email;
    return $res;
}

```

```

/**
 * Gets the currency code.
 * @return string A valid currency code
 */
public function GetCurrency() { return 'EUR'; }

```

```

/**
 * Gets the invoice ID.
 * @return mixed Invoice identifier
 */
public function GetInvoiceId() { return "I".$this->id; }

```

```

/**
 * Gets the order culture code.
 *
 * See <CultureInfo>
 * @return string Valid culture code
 */
public function GetLocale() { return 'en-US'; }

```

```

/**

```

```

    * Return the total price incl. VAT (if VAT applies for the given country).
    * @param float $price The price without VAT.
    * @return float Price including VAT (if VAT applies for the country).
    */
    public function GetTotalPrice($price = false)
    {
        if( $price !== false )
            return $price * ( (1+$this->GetVatPercent()) / 100 );
        return $this->price_total * ( (1+$this->GetVatPercent()) / 100 );
    }

    /**
     * Return the total VAT (if VAT applies for the given country).
     * @return float VAT in order currency
     */
    public function GetTotalVat() { return $this->price_total * ($this->GetVatPercent()/100); }

    /**
     * Return the total VAT percent (if VAT applies for the given country).
     * @return float VAT percent
     */
    public function GetVatPercent() { return 19; }

    /**
     * Returns all items.
     *
     * @return array A list of <IShopOrderItem> objects
     */
    public function ListItems() { return
SampleShopOrderItem::Make()->eq('order_id',$this->id)->orderBy('id'); }

    /**
     * Sets the currency
     * @param string $currency_code A valid currency code
     * @return void
     */
    public function SetCurrency($currency_code) { /* we stay with EUR */ }

    /**
     * Creates an instance from an order id.
     * @return IShopOrder The new/loaded order <Model>
     */
    public static function FromOrderId($order_id)
    {
        return SampleShopOrder::Make()->eq('id',$order_id)->current();
    }

    /**
     * Called when the order has failed.
     *

```

* This is a callback from the payment processor. Will be called when there was an error in the payment process.

* This can be synchronous (when customer aborts in then initial payment ui) or asynchronous when something goes wrong

* later in the payment processors processes.

* @param int \$payment_provider_type Provider type identifier

(<PaymentProvider>::PROCESSOR_PAYPAL, <PaymentProvider>::PROCESSOR_GATE2SHOP, ...)

* @param mixed \$transaction_id Transaction identifier (from the payment provider)

* @param string \$statusmsg An optional status message

* @return void

*/

public function SetFailed(\$payment_provider_type, \$transaction_id, \$statusmsg = false)

{

 \$this->status = self::FAILED;

 \$this->updated = \$this->deleted = 'now()';

 \$this->Save();

}

/**

* Called when the order has been paid.

*

* This is a callback from the payment processor. Will be called when the customer has paid the order.

* @param int \$payment_provider_type Provider type identifier

(<PaymentProvider>::PROCESSOR_PAYPAL, <PaymentProvider>::PROCESSOR_GATE2SHOP, ...)

* @param mixed \$transaction_id Transaction identifier (from the payment provider)

* @param string \$statusmsg An optional status message

* @return void

*/

public function SetPaid(\$payment_provider_type, \$transaction_id, \$statusmsg = false)

{

 \$this->status = self::PAID;

 \$this->updated = \$this->completed = 'now()';

 \$this->Save();

}

/**

* Called when the order has reached pending state.

*

* This is a callback from the payment processor. Will be called when the customer has paid the order but the

* payment has not yet been finished/approved by the provider.

* @param int \$payment_provider_type Provider type identifier

(<PaymentProvider>::PROCESSOR_PAYPAL, <PaymentProvider>::PROCESSOR_GATE2SHOP, ...)

* @param mixed \$transaction_id Transaction identifier (from the payment provider)

* @param string \$statusmsg An optional status message

* @return void

*/

public function SetPending(\$payment_provider_type, \$transaction_id, \$statusmsg = false)

{

 \$this->status = self::PENDING;

```

        $this->updated = 'now()';
        $this->Save();
    }

    /**
     * Called when the order has been refunded.
     *
     * This is a callback from the payment processor. Will be called when the payment was refunded
     for any reason.
     * This can be reasons from the provider and/or from the customer (when he cancels the payment
     later).
     * @param int $payment_provider_type Provider type identifier
     (<PaymentProvider>::PROCESSOR_PAYPAL, <PaymentProvider>::PROCESSOR_GATE2SHOP, ...)
     * @param mixed $transaction_id Transaction identifier (from the payment provider)
     * @param string $statusmsg An optional status message
     * @return void
     */
    public function SetRefunded($payment_provider_type, $transaction_id, $statusmsg = false)
    {
        $this->status = self::REFUNDED;
        $this->updated = $this->deleted = 'now()';
        $this->Save();
    }

    /**
     * Checks if VAT needs to be paid.
     * @return boolean true or false
     */
    public function DoAddVat() { return true; /* Let's assume normal VAT customers for now */ }
}

```

You will also need to create a class for order items, our is called Sample Shop Order Item and implements the IShop Order Item interface.

// model/sampleshoporderitem.class.php

```
<?php
```

```
use ScavixWDF\Model\Model;
```

```
use ScavixWDF\Payment\IShopOrderItem;
```

```
/**
```

```
 * Represents an order item in the database.
```

```
 *
```

```
 * In fact nothing more than implementations for the inherited Model
```

```
 * and the implemented IShopOrderItem interface.
```

```
 * See
```

```
https://github.com/ScavixSoftware/WebFramework/wiki/classes\_modules\_payment#wiki-97745ff2e14aebb2225c7647a8a059bc
```

```
 */
```

```
class SampleShopOrderItem extends Model implements IShopOrderItem
```

```
{
```

```
    /**
```

```
     * Returns the table name.
```


* See
https://github.com/ScavixSoftware/WebFramework/wiki/classes_essentials_model_model.class#gettablename

```
*/  
public function GetTableName() { return 'items'; }  
  
/**  
 * Gets the price per item converted into the requested currency.  
 * @param string $currency Currency code  
 * @return float The price per item converted into $currency  
 */  
public function GetAmount($currency) { return $this->price; }  
  
/**  
 * Gets the discount.  
 * @return float The discount  
 */  
public function GetDiscount() { return 0; }  
  
/**  
 * Gets the handling cost.  
 * @return float Cost for handling  
 */  
public function GetHandling() { return 0; }  
  
/**  
 * Gets the items name.  
 * @return string The item name  
 */  
public function GetName() { return $this->title; }  
  
/**  
 * Gets the quantity.  
 * @return float The quantity  
 */  
public function GetQuantity() { return $this->amount; }  
  
/**  
 * Gets the shipping cost.  
 * @return float Cost for shipping  
 */  
public function GetShipping() { return 0; }  
}
```

Finally the payment module has to be configured:

// config.php

<?php

// full code: https://github.com/ScavixSoftware/WebFramework/blob/master/web/sample_shop/config.php

// configure payment module with your IShopOrder class

\$CONFIG["payment"]["order_model"] = 'SampleShopOrder';

```
// set up Gate2Shop if you want to use it
$CONFIG["payment"]["gate2shop"]["merchant_id"] = '<your_merchant_id>';
$CONFIG["payment"]["gate2shop"]["merchant_site_id"] = '<your_merchant_site_id>';
$CONFIG["payment"]["gate2shop"]["secret_key"] = '<your_secret_key>';
// set up PayPal if you want to use it
$CONFIG["payment"]["paypal"]["paypal_id"] = '<your_paypal_id>';
$CONFIG["payment"]["paypal"]["notify_handler"] = array('Basket','Notification');
```

The Administration

For a shop system you will need to be able to create products and have some kind of access to your customers data and their orders.

Once you enter the administrative page it will ask you for credentials and (as mentioned above) they are hardcoded: use 'admin' as username and 'admin' as password.

```
// controller/admin.class.php
<?php
use ScavixWDF\Base\AjaxAction;
use ScavixWDF\Base\AjaxResponse;
use ScavixWDF\Base\Template;
use ScavixWDF\Controls\Form\Form;
use ScavixWDF\jQueryUI\Dialog\uiDialog;
use ScavixWDF\jQueryUI\uiButton;
use ScavixWDF\jQueryUI\uiDatabaseTable;
use ScavixWDF\jQueryUI\uiMessage;

class Admin extends ShopBase
{
    /**
     * Checks if aa admin has logged in and redirects to login if not.
     */
    private function _login()
    {
        // check only the fact that somebody logged in
        if( $_SESSION['logged_in'] )
            return true;

        // redirect to login. this terminates the script execution.
        redirect('Admin','Login');
    }

    /**
     * @attribute[RequestParam('username','string',false)]
     * @attribute[RequestParam('password','string',false)]
     */
    function Login($username,$password)
    {

```

```

        // if credentials are given, try to log in
        if( $username && $password )
        {
            // see config.php for credentials
            if( $username==cfg_get('admin','username') &&
$password==cfg_get('admin','password') )
            {
                $_SESSION['logged_in'] = true; // check only the fact that somebody
logged in
                redirect('Admin');
            }
            $this->content(uiMessage::Error("Unknown username/passsword"));
        }
        // putting it together as control here. other ways would be to create a new class
        // derived from Control or a Template (anonymous or with an own class)
        $form = $this->content(new Form());
        $form->content("Username:");
        $form->AddText('username', "");
        $form->content("<br/>Password:");
        $form->AddPassword('password', "");
        $form->AddSubmit("Login");
    }
}

```

/* full code:

```

https://github.com/ScavixSoftware/WebFramework/blob/master/web/sample\_shop/controller/admin.class.
php*/
}

```

Now we will start with the first part mentioned above: the product management.

// controller/admin.class.php

<?php

```

use ScavixWDF\Base\AjaxAction;
use ScavixWDF\Base\AjaxResponse;
use ScavixWDF\Base\Template;
use ScavixWDF\Controls\Form\Form;
use ScavixWDF\jQueryUI\Dialog\uiDialog;
use ScavixWDF\jQueryUI\uiButton;
use ScavixWDF\jQueryUI\uiDatabaseTable;
use ScavixWDF\jQueryUI\uiMessage;

```

class Admin extends ShopBase

{

/* full code:

```

https://github.com/ScavixSoftware/WebFramework/blob/master/web/sample\_shop/controller/admin.class.
php*/

```

function Index()

{

 \$this->_login(); // require admin to be logged in

 // add products table and a button to create a new product

```

$this->content("<h1>Products</h1>");
$this->content(new uiDatabaseTable(model_datasource('system'),false,'products'))
    ->AddPager(10)
    ->AddRowAction('trash', 'Delete', $this, 'DelProduct');
$this->content(uiButton::Make('Add product'))->onclick = AjaxAction::Post('Admin',
'AddProduct');

// add orders table
$this->content("<h1>Orders</h1>");
$this->content(new uiDatabaseTable(model_datasource('system'),false,'orders'))
    ->AddPager(10)
    ->OrderBy = 'id DESC';

// add customers table
$this->content("<h1>Customers</h1>");
$this->content(new uiDatabaseTable(model_datasource('system'),false,'customers'))
    ->AddPager(10)
    ->OrderBy = 'id DESC';
}

/**
 * @attribute[RequestParam('title','string',false)]
 * @attribute[RequestParam('tagline','string',false)]
 * @attribute[RequestParam('body','text',false)]
 * @attribute[RequestParam('price','double',false)]
 */
function AddProduct($title,$tagline,$body,$price)
{
    $this->_login(); // require admin to be logged in

    // This is a quite simple condition: You MUST provide each of the variables
    if( $title && $tagline && $body && $price )
    {
        // store the uploaded image if present
        if( isset($_FILES['image']) && $_FILES['image']['name'] )
        {
            $i = 1; $image = __DIR__.'../images/'.$_FILES['image']['name'];
            while( file_exists($image) )
                $image =
__DIR__.'../images/'.$i++.'_'.$_FILES['image']['name'];
            move_uploaded_file($_FILES['image']['tmp_name'], $image);
            $image = basename($image);
        }
        else
            $image = "";

        // store the new product into the database
        $ds = model_datasource('system');
        $ds->ExecuteSql("INSERT INTO
products(title,tagline,body,image,price)VALUES(?,?,?,?,?)",
array($title,$tagline,$body,$image,$price));

```

```

        redirect('Admin');
    }
    // create a dialog and put a template on it.
    $dlg = new uiDialog('Add product',array('width'=>600,'height'=>450));
    $dlg->content( Template::Make('admin_product_add') );
    $dlg->AddButton('Add product', "$(#frm_add_product).submit()"); // frm_add_product
is defined in the template
    $dlg->AddCloseButton("Cancel");
    return $dlg;
}

/**
 * @attribute[RequestParam('table','string',false)]
 * @attribute[RequestParam('action','string',false)]
 * @attribute[RequestParam('model','array',false)]
 * @attribute[RequestParam('row','string',false)]
 */
function DelProduct($table,$action,$model,$row)
{
    $this->_login(); // require admin to be logged in

    // we use the ajax confirm features of the framework which require some translated string,
so we set them up here
    // normally we would start the sysadmin and create some, but for this sample we ignore
that.
    default_string('TITLE_DELPRODUCT','Delete Product');
    default_string('TXT_DELPRODUCT','Do you really want to remove this product? This
cannot be undone!');
    if( !AjaxAction::IsConfirmed('DELPRODUCT') )
        return AjaxAction::Confirm('DELPRODUCT', 'Admin', 'DelProduct',
array('model'=>$model));




    // load and delete the product dataset
    $ds = model_datasource('system');
    $prod = $ds->Query('products')->eq('id',$model['id'])->current();
    $prod->Delete();

    // delete the image too if present
    if( $prod->image )
    {
        $image = __DIR__.'../images/'.$prod->image;
        if( file_exists($image) )
            unlink($image);
    }
    return AjaxResponse::Redirect('Admin');
}
}

```


SCREEN SHOT

The products listing page:

| Products | Basket | Administration (normally hidden) |
|---|--|--|
|  | Product 1
This is short desc for product 1 | Details... |
|  | Product 2
Product 2 has a tagline too | Details... |
|  | Product 3
Product 3 tagline: we need that for listings | Details... |

The product's details page:

[Products](#) [Basket](#) [Administration \(normally hidden\)](#)





Product 3

No desc here too as this is demo data

[Add to basket](#) [back to listing](#)

That's it for the listing/editing part of the basket. The result looks like this:

| Products | Basket | Administration (normally hidden) |
|---|---|---|
|  | Product 3 Amount: 1 Price: 1 99 Total: 1 99 | add one more remove one |
|  | Product 2 Amount: 1 Price: 9 85 Total: 9 85 | add one more remove one |
| Buy now | | Total price: 11 84 |

which payment provider he wants to use:

[Products](#) [Basket](#) [Administration \(normally hidden\)](#)

Address:

| | | |
|-------------------|--|---------------------------------------|
| Firstname | <input type="text" value="Ajay"/> | |
| Lastname | <input type="text" value="Singh"/> | |
| Street and number | <input type="text" value="Bakerstr. 222"/> | |
| ZIP and City | <input type="text" value="12345"/> | <input type="text" value="Overhere"/> |
| E-Mail address | <input type="text" value="my@domain.com"/> | |

Payment provider:

- ☐ PayPal
☐ Gate2Shop
☒ Testing

'DelProduct' when clicked:

[Products](#) [Basket](#) [Administration \(normally hidden\)](#)

Products

| id | title | tagline |
|----|-----------|--------------------------------|
| 1 | Product 1 | This is short d |
| 2 | Product 2 | Product 2 has |
| 3 | Product 3 | Product 3 tagl
for listings |

Add product

Add product

TitleNew Product

Tagline

Description

ImageBrowse...

Price

Add product

Cancel

Orders

| id | customer_id | status | c |
|----|-------------|--------|---|
| 1 | 1 | 20 | 2 |

Customers

| id | email | fname | lname | street | zip | city |
|----|---------------|--------|-------|---------------|-------|----------|
| 1 | my@domain.com | Daniel | Spors | Bakerstr. 222 | 12345 | Overhere |

[Products](#) [Basket](#) [Administration \(normally hidden\)](#)

Products

| id | title | tagline | body | image | price |
|----|-----------|--|--|--------------|-------|
| 1 | Product 1 | This is short desc for product 1 | Here we go with an real description that can be long and will only be displayed on products details page | product1.png | 11.99 |
| 2 | Product 2 | Product 2 has a tagline too | But we will go with a short description | product2.png | 9.85 |
| 3 | Product 3 | Product 3 tagline: we need that for listings | No desc here too as this is demo data | product3.png | 1.99 |

Add product

Orders

| id | customer_id | status | created | updated | completed | deleted | price_total |
|----|-------------|--------|---------------------|---------------------|---------------------|---------|-------------|
| 1 | 1 | 20 | 2013-05-06 10:24:48 | 2013-05-06 10:24:48 | 2013-05-06 10:24:48 | | 11.84 |

Customers

| id | email | fname | lname | street | zip | city |
|----|---------------|--------|-------|---------------|-------|----------|
| 1 | my@domain.com | Daniel | Spors | Bakerstr. 222 | 12345 | Overhere |

RESULT AND ADVANTAGES

INTERPRETATION OF THE RESULT

The system has been implemented and tested successfully. It meets the information Requirements specified to the great extent. Although the system has been designed keeping the

Present and future requirements in mind and made very flexible. There are limitations of the

System. Proper consideration has been given for a wide range of new enhancements in

The future, through out the development of system. The system is developed user friendly. In future, if it is required to generate reports other than provided by the system, it can be simply

Achieved by a separate module to the main menu without affecting the design of the system.

ADVANTAGES

- It simplifies the operation.
- It avoids a lot of manual work.
- Every Transaction is obtained and processed immediately.
- Avoids errors by avoiding the manual work.
- User friendly screen to enter the data and Enquire the database tables.
- Online help messages available to the operating system.
- User can easily access the system without much experience.
- Provide Hardware and software securities.
- Portable and flexible for further extension.

CONCLUSION

The central concept of the application is to allow the customer to shop virtually using the Internet and allow customers to buy the items and articles of their desire from the store. The information pertaining to the products are stores on an RDBMS at the server side (store). The Server process the customers and the items are shipped to the address submitted by them.

The application was designed into two modules first Os for the customers who wish to buy the articles. Second is for the storekeepers who maintains and updates the information pertaining to the articles and those of the customers?

The end user of this product is a departmental store where the application is hosted on the web and the administrator maintains the database. The application which is deployed at the customer database, the details of the items are brought forward from the database for the customer view based on the selection through the menu and the database of all the products are updated at the end of each transaction.

Data entry into the application can be done through various screens designed for various levels of users. Once the authorized personnel feed the relevant data into the system, several reports could be generated as per the requirements.

This system offers information relevant to the user accessing the application thus avoiding unnecessary overloading and at the same time maintaining the security.

BIBLIOGRAPHY

HTML Publishing Bible - Alan Simpson.

Netscape Java Script - Peter & John Kent.

The Complete Reference Of Java - Patrick Naughton

Java Network programming - Rusty Harod.

Software Engineering - Fairly.

Analysis & Design
Of Information - James A. Senn.