



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

School of Applied Computing

Student Name:	Peter Davies	Student P number:	P079868
Programme:	BSc Web Development		
Module:	Distributed Enterprise Applications	Lecturer:	Dr Tim Bashford
Assignment Title:	Retailer WCF		
Hand Out Date:	February 2019	Submission deadline:	08/05/19

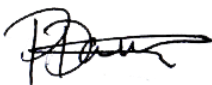
Disclosure:	I declare that this assignment is all my own work and that I have acknowledged all materials used from the published or unpublished works of other people. All references have been duly cited.		
Student's Signature:		Date:	08/05/19

Table of Contents

1. Introduction	1
2. WCF Hosted Service.....	2
3. SQL Database.....	5
3.1 ADO.NET	5
4. .NET MVC Client	6
4.1 Client Components & Features.....	6
4.1.1 Shopping Cart.....	6
4.1.2 User Authentication	9
5. Conclusion.....	12
5.1 Objectives Met	12
6. Reflective Evaluation.....	14
7. References	15
8. Bibliography	16

1. Introduction

The following report critically analyses the key features implemented in the e-commerce application produced for the sale of water sport goods.

It firstly documents the WCF service implemented, and the database which the WCF service communicates followed by analysis of the .NET MVC client used to consume the WCF service, and the features it implements.

Finally, the assignment is concluded and reflected on. A full reference list and bibliography ensues.

NB. The application provided uses Git version control. The latest branch (ViewModel) contains code which attempted to implement a view model to pass to the _Layout view, however, this results in errors in code which prevent the functionality of the application, therefore, please run the code on the SimulateCheckout branch which has been tested to work.

2. WCF Hosted Service

The WCF service was created using Microsoft Visual Studio's WCF template [1], which includes the necessary files required to build and host WCF services; the service contract file, an interface which defines the different types of contract used, the service implementation file, which inherits off the service contract file, allowing for definition of the service behaviours and the *Web.Config* file containing the basic elements needed, namely, a secure *HTTPBinding*, a service endpoint declaration, and enabling of metadata exchange which exposes data about the service to the client [2]. These files are known as the ABCs of the WCF [3].

The WCF service is used in this instance to expose data for an e-commerce web application's data store, allowing customers to browse items, add items to a cart and register their details to simulate a checkout over a network. The simple database structure is shown in figure 1 in section 2.1.

To enable the client (section 3) to consume this WCF service, it must provide an endpoint. This endpoint defines the three vital pieces of information required to act as the gateway to the service [4]. These 3 pieces of information, as discussed earlier are the ABCs of the service. The ABCs of the service created are:

The Address (A) – the address where the service exists. This is the address (URL) the client will use to connect to the service. It is declared declaratively in the *Web.Config* file in both the WCF service and the .NET client, as this service is hosted locally, the service points to the localhost.

```
<endpoint address="http://localhost:51311/Service.svc"
```

The Binding (B) – which defines how the communication with the service will take place. The basic binding was used for this application as it provides the greatest interoperability, however, given the client is a .NET application which required high security levels when transferring data through the service, on reflection, a more secure binding such as *wsHttp* could have been chosen.

```
binding="basicHttpBinding"
```

The Contract (C) – the contract specifies what the endpoint created will communicate to the client [5]. The service created has three contracts;

- **Service Contract** – the interface of the service, specifying what the service will do. The service contract is the *IService* interface.

```
[ServiceContract]
1 reference | Peter Davies, 22 hours ago | 1 author, 9 changes
public interface IService
{
}
```

- **Operation Contract** – included in the service contract as decorators to the defined interface methods, the operation contract indicates what each method does by stating the parameters and return types [6]. A sample of the operation contracts used is shown below.

```
[ServiceContract]
1 reference | Peter Davies, 22 hours ago | 1 author, 9 changes
public interface IService
{
    [OperationContract]
    1 reference | Peter Davies, 3 days ago | 1 author, 1 change
    List<Items> GetAllItems();

    [OperationContract]
    1 reference | Peter Davies, 2 days ago | 1 author, 2 changes
    Items GetItemById(int id);

    [OperationContract]
    1 reference | Peter Davies, 3 days ago | 1 author, 1 change
    int DeleteItemById(int id);
}
```

- **Data Contract** – again defined as a decorator, these contracts define the data to be passed between the service and the client through the properties, decorated with the *DataMember* attribute, which match the database entities, Models used in MVC.

```
[DataContract]
10 references | Peter Davies, 2 days ago | 1 author, 2 changes
public class Items
{
    [DataMember]
    2 references | Peter Davies, 3 days ago | 1 author, 1 change
    public int ItemId { get; set; }

    [DataMember]
    2 references | Peter Davies, 3 days ago | 1 author, 1 change
    public string ItemName { get; set; }
}
```

The *IService* interface, or *ServiceContract* file contains the interface methods along with all *DataContracts* which define the data each service will use. It is considered that the best practice would be to move these files into a separate file location as should the application be scaled up; this interface would become very bulky and difficult to maintain.

The service implementation file, defines the behaviour of each *OperationContract* method, in the example below, the method retrieves the data from the database using a *Linq* query, iterates through the returned results and setting the *DataMember* property values in the *Items* class before finally adding each row's properties to a list which will be communicated over the service for consumption by the client.

```
1 reference | Peter Davies, 1 day ago | 1 author, 3 changes
public List<Items> GetAllItems()
{
    List<Items> items = new List<Items>();
    SFSSurfEntities sfs = new SFSSurfEntities();
    var itemList = from k in sfs.Items select k;
    foreach (var item in itemList)
    {
        Items product = new Items();
        product.ItemId = item.ItemId;
        product.ItemName = item.ItemName;
        product.ItemPrice = item.ItemPrice;
        product.ImageName = item.ImageName;
        product.ItemDesc = item.ItemDesc;
        items.Add(product);
    }
    return items;
}
```

The service class defines the behaviour of all the interfaces methods which will be communicated over the WCF service.

3. SQL Database

The data to be exposed by the service is contained in an SQL database consisting of three tables, one to store the products on sale, another to store products added to the cart by customers and one to handle the user authentication.

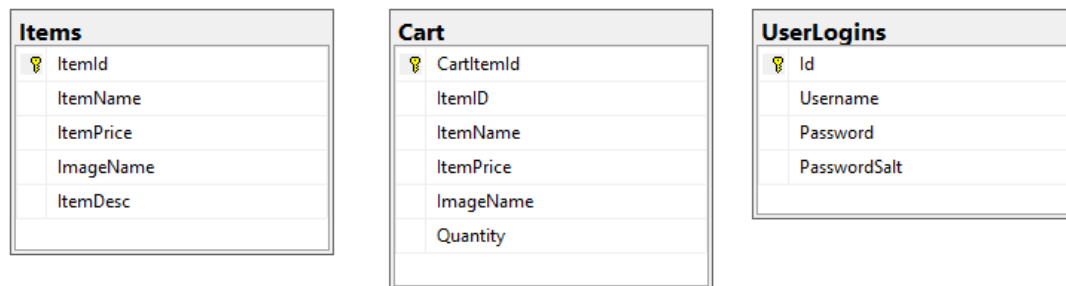


Figure 1 - Entity Relationship Diagram

The *Cart* table contains similar data to the *Items* table, and it is best practice to have a foreign key relationship between these tables on the *ItemId* field, however, when querying this table over the service, an exception occurred due to this constraint. With no time to investigate, it was removed.

It is understood that the database implemented is very simple and does not contain specific information required for an e-commerce application, such as, names and addresses of customers and card payment details. This is due to the limited time frame of which this application was produced. The tables were implemented to demonstrate simple CRUD functionality over a WCF service. The SQL files will accompany the code submission of the report.

3.1 ADO.NET

To map the database to the WCF service the ADO.NET data access tool is used to provide a mapping-driven data access layer [7] which allows the database to be queried using objects in the programming language (C#).

4. .NET MVC Client

The client chosen to consume the WCF service was a .NET MVC web application installed using the visual studio template which contained all the relevant files and application file structure [7]. It is created inside the same project solution as the WCF service with a service reference to the WCF service added to the MVC app. As the two projects reside in the same solution, the WCF service was set to start before the MVC app to enable the app to communicate with the service immediately.

4.1 Client Components & Features

4.1.1 Shopping Cart

The application allows users to browse products, data-driven from the *Items* database table, displayed in the store section of the application. Here users can add items to a cart.



Cart						
Image	Item Name	Item Price	Quantity	Update Quantity	Remove from Cart	Total
	FCS Performer Fins	£84.99	1	Update Quantity »	Remove from Cart »	84.99
	JP RX Surfboard	£599.99	1	Update Quantity »	Remove from Cart »	599.99

The products are added to the cart via a .NET Html helper *ActionLink* on the products page, which passes in the details of the item to the *CartController* method *AddItem*.

```
<p class="card-text"><b>£@item.ItemPrice</b></p>
<div class="btn-links btn btn-success">@Html.ActionLink("Add to Cart ", "AddItem", "Cart", new { ItemID = item.ItemId, Item
```

Items in the cart can then be removed using the *Remove from Cart* button which again uses a Html helper to call the correct method in the *CartController* (*DeleteItem*) which contains a query to remove the item from the *Cart* table.

An additional feature of the cart is the customers ability to update the quantity of items and the total amount of the products being updated in the far-right column of the table. This feature proved to be more challenging to implement than first envisaged.

The products in the cart are displayed by using a *foreach* loop to iterate through the items in the *Cart* table, with a form added allowing for a text box to be displayed for customers edit the total amount of an item.

```
@foreach (var prod in Model)
{
    <tr>
        <th scope="row"> prod.CartItemId);
            @Html.HiddenFor(model => prod.ItemID);
            @Html.HiddenFor(model => prod.ItemName);
```

However, upon testing of this feature the cart contents were not updating. To investigate this a breakpoint was added to the *UpdateQuantity* method in the *CartController*, which discovered that the data from the form was not being passed to the controller's method.

```

64
65
66
67 // Having to pass in all the properties as some
68 ShoppingCart cartQ = new ShoppingCart();
69 cartQ.CartItemId = cart.CartItemId;
70 cartQ.ItemID = cart.ItemID;
71 cartQ.ItemName = cart.ItemName;
72 cartQ.ItemPrice = cart.ItemPrice;
73 cartQ.Quantity = cart.Quantity;
74 int? save = sc.UpdateQuantity(cartQ, cart.Quantity | null);
75 if(save > 0)

```

cart	(SF5MVC.Model)
CartItemId	0
GrandTotal	null
ImageName	null
ItemID	0
ItemName	null
ItemPrice	0
Quantity	null
Total	null

Further investigation led to the notion that within a .NET foreach loop, in a view, the HTML form which is generated by the Html form helper method does not add the name attribute to the input meaning that the data from the form cannot bind to the property to be passed to the controller action.

To overcome this problem, a *for* loop was used instead with the count set to the number of items in the *Cart* table. This would then generate the correct HTML table required. The name attribute was then added using the *@Name* property on each field. The form now submits the data, correctly bound to the model to be executed by the controller's method.

```
@for (int i = 0; i < cart.Count(); i++)
{
    <tr>
        <th scope="row"> cart[i].CartItemId, new { @Name = "CartItemId" });
        @Html.HiddenFor(model => cart[i].ItemID, new { @Name = "ItemID" });
        @Html.HiddenFor(model => cart[i].ItemName, new { @Name = "ItemName" });
        @Html.HiddenFor(model => cart[i].ItemPrice, new { @Name = "ItemPrice" });
    }
}
```

This problem with the cart implementation was one of many. For example, when adding an item to the cart the page should automatically redirect to the cart, however, unlike other redirects used throughout this application, this did not work. Furthermore, if the same product is added to the cart then it appears twice in the cart, instead of the quantities of that item being updated, something which could be remedied by modifying the query.

Additionally, the grand total logic is wrong, calculating the totals for each row multiplying then by the count, and as it is implemented inside the *for* loop, it repeats and is displayed twice for each cart row. And finally, as the cart contents are stored in a database table, the cart does not clear when a session ends, meaning when a new user logs in, the previous user's cart contents are still displayed.

4.1.2 User Authentication

The application allows for users to login to the system by matching usernames and password in the database on the server side using the WCF service.

```
public string UserLogin()
{
    var login = (from user in sc.UserLogins where user.Username == Username && user.Password == Password select user).Count() > 0;
    if (!login)
    {
        return "false";
    }
    else
    {
        return "true";
    }
}
```

This method is then called inside the *LoginController* and the returned result checked to enable entry to the system. If "true" is returned the username logged in is stored in the session variable and the user redirected to the home page. If the usernames or passwords do not match records in the database, then the user is alerted with a simple JavaScript popup.

```
public ActionResult Login(SFSServiceReference.ULogins log)
{
    string check = sc.UserLogin(log);
    if (check == "true")
    {
        Session["Username"] = log.Username.ToString();
        //System.Diagnostics.Debug.WriteLine(Session["Id"]);
        return RedirectToAction("Index", "Home");
    }
    else
    {
        TempData["error"] = "<script> alert('Incorrect Username or Password');";
        return View();
    }
}
```

The session variable is used to deny access to certain areas of the application and display certain links in the header by using *if-else* blocks to check if there is anything stored in the session variable. If it is *null* the user is asked to login, otherwise the correct page is displayed.

```
<h2>Checkout</h2>
@if (Session["Username"] == null)
{
    <strong>Please Log in!</strong>
    <div class="sign-up">
        <p>Please Log in @Html.ActionLink("here", "Login", "Login") to checkout. Not yet a regis
    </div>
}
else
{
    <fieldset>
        <legend> Checkout </legend>
        <br />
        <strong>
            Welcome @Session["Username"].ToString()
        </strong>
    </fieldset>
}
```

The login system implemented is very simple and insecure. Passwords should never be stored as plain text directly in a database or in an encrypted format which could be decrypted, it is best practice to hash the passwords using a one-way function and provide a salt key to add an additional input to protect against dictionary or rainbow attacks.

It was intended to implement this security feature into the application, evidence is seen in the *UserLogins* table, however, limited time prevented this as hashing passwords is not something which has been investigated or implemented in .NET before. A workaround this would have been to use the built in .NET Identity feature which handles password hashing or external logins, as well as allowing for 'roles' which allow users different levels of access to an application's features. Identity has been used before, however, implementation over a WCF service has not, therefore it was omitted as no documentation could be found relevant to this problem.

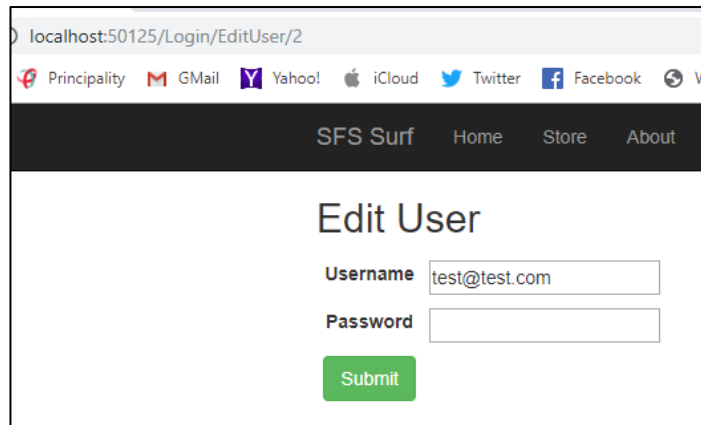
The register form also suffers from simplicity as it provides no validation, in terms of errors to email entry or password format, for example. Furthermore, it allows for duplicate entries to be registered.

4.1.2.1 Updating User Details

Logged in users can update their details. Using similar forms and database queries discussed in section 4.1.1. To access their 'account' area to update their details, a link was added in the shared layout view, however, as this layout is shared across every page in the application it does not have a model associated with it, meaning that there was no way to pass the current users ID to the *EditUser* method in the *LoginController*, which results in the system breaking as the Model's value is null.

```
<li>@Html.ActionLink("Log Out", "Logout", "Login")</li>
<li>@Html.ActionLink("Account", "EditUser", "Login")</li> // NEED TO PASS IN THE CURRENT USER ID
```

The solution to this problem would be to create a new view model with the properties required along with a base controller to allow these properties to be passed to the layout view. This was quickly attempted, however, it was unable to be implemented due to lack of time, therefore, should the user wish to update their details the only workaround is to enter the URL with their ID passed in.



Something which is obviously incorrect, furthermore, the security implemented, or lack of, means that any user can simply enter this URL with anyone's ID and edit their details. Again, lack of time meant that these features were extremely rushed, and not enough time was permitted for investigation of correct implementation.

5. Conclusion

This assignment asked the difficult task of implementing an e-commerce application using a WCF service, something which has proved very challenging. The solution provided to this problem does address parts of the assignment brief, in terms of hosting a WCF service and providing a simplistic attempt at an e-commerce application, however, there are many areas which need improving.

The implementation of the shopping cart proved particularly challenging, as features of this nature have never been developed before, meaning immense unfamiliarity with the logic and approach needed to successfully develop a workable solution. Due to limited time to work out the logic, and lack of pertinent documentation to research this area properly it can be concluded the cart produced is poor.

Additionally, the database design is very simple, but has been designed in this way to be able to deliver *something* to assess for the second half of this module. Again, more time would have meant a correctly structured, relational and normalised database. Furthermore, the assignment brief emphasised a focus on security, however, this application is far from secure, but this is something which is acknowledged, and suggestions have been provided into better security practices.

In conclusion, the application provides an extremely basic attempt at delivering a commercial distributed system, therefore it is not recommended that it could be deployed without considerable re-development.

5.1 Objectives Met

Despite the application produced being poor and needing considerable re-development to be used commercially, it has, albeit very basically, met the following objectives:

Must Have

- Register users and allow them to login to the system.
- Ensure that only logged on users can purchase items.
- Permit the purchase of multiple or duplicate items in a single transaction.

Should Have

- Allow users to modify their own details.

6. Reflective Evaluation

Reflecting on this assignment has proved uncomfortable, as it is far below the standards I expect and have set for myself over my academic career at UWTSD.

It is very disappointing that I am submitting an assignment not up to standards, however, I feel that given more time to complete this assignment I would have taken an entirely different approach. With the final year project due in 1st May, and work commitments on 2nd and 3rd May, followed by a bout of illness 4th and 5th May, this left me with three days to complete this assignment, resulting in the correct approach and research not being carried out, in turn resulting in a sub-standard submission. However, I do not feel the previous statement should be an excuse for not achieving my best, although, the certain factors mentioned have played their part. Better time management overall in my final year would have resulted in a more consistent standard of work in this module.

Despite this, I do feel that I have delivered on some of the objectives set, and, I feel web development teams would struggle to implement every feature given in the assignment brief professionally, in the time allocated for development, even without the added pressure of other university commitments.

7. References

- [1] “WCF Visual Studio Templates | Microsoft Docs,” *Microsoft Developer Network*, 2017. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/framework/wcf/wcf-vs-templates>. [Accessed: 08-May-2019].
- [2] J. Lowy, *Programming WCF Services*, 3rd Editio. O’Reilly, 2010.
- [3] H. Cheng, “WCF Windows Communication Foundation.”
- [4] C. Peiris, D. Mulder, and A. Com, *Pro WCF - Practical Microsoft SOA Implementation*. Apress, 2007.
- [5] “Contracts | Microsoft Docs,” *Microsoft Developer Network*, 2017. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/framework/wcf/feature-details/contracts>. [Accessed: 22-Jan-2019].
- [6] “OperationContractAttribute Class (System.ServiceModel) | Microsoft Docs,” *Microsoft Developer Network*, 2017. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.operationcontractattribute?view=netframework-4.7.2>. [Accessed: 22-Jan-2019].
- [7] “Getting Started with ASP.NET MVC 5 | Microsoft Docs,” *Microsoft Developer Network*, 2018. [Online]. Available: <https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/getting-started>. [Accessed: 08-May-2019].

8. Bibliography

M. L. Bustamante, "Hosting WCF Services," *CODE Magazine*. [Online]. Available: <https://www.codemag.com/article/0701041/Hosting-WCF-Services>. [Accessed: 19-Jan-2019].

P. R. Egli, "WCF-Windows Communication Foundation," 2015.

Microsoft, "Windows Communication Foundation."

S. Klein, "WCF Programming .NET Development with the Windows ® Communication Foundation," 2007.

N. Pathak, *Pro WCF 4 Practical Microsoft SOA Implementation*, 2nd ed. Apress, 2011.

M. L. Bustamante, *Learning WCF*. O'Reilly, 2007.

S. Cheng, *Microsoft Windows Communication Foundation 4.0 Cookbook for Developing SOA Applications*. Birmingham, Mumbai: Packt Publishing, 2010.

J. Sharp, *Windows Communication Foundation 4 Step by Step*, vol. 23. 2010.