Application design of a layered architecture in ASP.NET 2.0

- and a comparison between J2EE and .NET

4-week project
IT-University of Copenhagen (ITU)
Handed in May 29th 2006

Supervisor:

Peter Sestoft

Authors:

Nils Kasper Emil Oldby
Mikkel Byrsø Dan

1 Introduction

This report has been written in a four-week project carried out as a part of the "C# and .NET project cluster May 2006" at the IT-University of Copenhagen. Further information about the project cluster is available at the project cluster's homepage[33]. The project was supervised by Peter Sestoft, who we thank for taking the time to supervise and for the valuable comments and feedback in the process of developing the sample application and writing this report.

1.1 Motivation

Our motivation for doing this project follows the main goal of the project cluster, namely to get a chance to familiarize ourselves with the C# programming language and the .NET class libraries, including the version 2.0 features released with Visual Studio 2005.

Furthermore we are interested in the design of object-oriented distributed applications and have some knowledge of the architecture and implementation of such applications in the J2EE framework. In general we are eager to learn about systems and architectures that are easy maintainable, scalable, flexible and that can exchange information with other systems and clients on other platforms in an internet context. Therefore this project cluster seems like the perfect combination for us, since it will allow us to look into the design of the type of applications just mentioned using the C# programming language and ASP.NET 2.0.

1.2 Problem definition and scope

The purpose of this project is to discuss architecture and application design of object-oriented distributed applications in general and in relation to the .NET platform. In addition we will try to elaborate on the differences between J2EE and .NET in relation to these subjects.

We will try to challenge the typical 2-tier application design that is encouraged in .NET and Visual Studio, where it's normally to make use of direct data binding from the user interface to an underlying data base. This is specifically possible because of the drag 'n' drop facilities in Visual Studio which makes development fast and easy but might have negative effects in relation to maintenance etc.

The project will discuss n-tier architecture from a theoretical perspective and examine how and what differences there are between n-tier application design using J2EE and .NET. The .NET design should not only be examined on a theoretical level but also be implemented. The implementation of a small sample application should illustrate blueprints of a .NET-based layered application architecture.

In the project an effort will also be placed on learning and using new features in ASP.NET 2.0 and Visual Studio 2005. Due to the fact that this project is written in a C# project cluster, the sample application will only be developed using C# / ASP.NET 2.0 and not using J2EE.

1.3 How this report is organized

This report could be viewed as consisting of three logical parts. The parts being, application design in general and in .NET, a sample application designed using .NET and finally a comparison between J2EE and .NET.

Each main chapter of the report will always include a short introduction. The purpose of these introductions is to briefly introduce the coming information and to keep the reader aware of where in the context the information fits.

Citations

The citation style used in this report is “bibliography style plain”, which means that we use square bracket parenthesis and a number when referring to a resource e.g. [1]. Our bibliography is as usual placed in the end of the report. It is arranged by the type of source e.g. books or online articles. Within each type the references are sorted alphabetically.

Computer code

Computer code is marked using a monospace font. Computer code looks like the following:

```
localhost.Service wS = new localhost.Service();
```

2 Application Design

The purpose of this section is to examine general architectural principles that exist in application design. Further it should be determined what advantages dividing an application into layers and tiers have. Finally different layered architectures will be described and their advantages examined. The overall purpose of this section is to get a general understanding of layered application design.

2.1 General design principles

In general, when designing applications it is important to focus on the system design in order to ease maintainability, interoperability, scalability, further development etc. One thing that is especially important is low coupling, hence it is important to "reduce the impact of change" [5, p. 285]. Briefly, coupling is a measure of how strong one element is connected to other elements, if it has knowledge of or depends on them. Elements in this context can be of all levels of granularity from classes in an object-oriented language to components to large sub-systems or software modules. If there is a high coupling or dependency between elements, then a change in the depended-upon element will cause a change in the dependant and this is often not desirable.

Another important matter is high cohesion, "how to keep objects focused, understandable, and manageable, and as a side-effect, support low coupling" [5, p. 315]. The term cohesion is a measure of how strongly and focused the responsibilities of an element are. Thus, an element with highly related responsibilities that does not do a lot of different tasks has high cohesion. High cohesion makes the system easier to comprehend, easier to reuse and maintain which in general is desirable.

2.2 Layered architecture

When the different elements in a system are organized systematically we talk about the system architecture. The architecture is the large-scale division of a system into layers or tiers, each with responsibility for a major part of the system and with as little direct influence on other layers as possible. This definition was derived from [11].

2.2.1 Logical and physical tiers

A tier can both refer to a logical and a physical entity. The term tier was originally introduced as meaning a logical layer [5, p. 207], but now has both a logical and a physical meaning because of the widespread client/server model and distributed applications.

The physical meaning of the tier term refers to the fact that different parts of a system can be placed on different hardware, e.g. client/server. In this case you can for instance place the user interface on the client and the business logic and data on the server. This is called a two-tiered architecture. In this situation you work with different communication protocols to exchange information between the physically divided tiers. A division into physical tiers is illustrated in Figure 1.

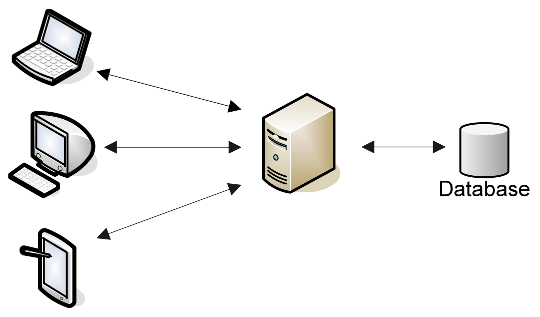


Figure 1 - Physical tiers

The logical meaning of the layer/tier term refers to a conceptually distinct aspect of a system e.g. separating the user interface from the business logic. The user interface can be implemented with different technologies and have different appearance, while the business logic remains the same. These two aspects are thus logically distinct and can be said to represent two distinct layers or tiers. Only tiers right beside each other communicate directly and they perform different tasks in the system as the example with the user interface and business logic.

2.2.2 Vertical division into layers

Separating the system into vertical layers means that the system is divided into layers or slices on top of each other. The higher layers call upon services from lower layers and not vice versa. It is also often ensured that all communication between layers is placed in distinct interaction classes and that each layer has a well-known interface so that it is always clear where and how the layers communicate.

In a vertical layer split-up different layers often provide the same functionality, just at a different abstraction level. This is opposite of the logical layer/tier, where the different layers provided different functionality. One example of a vertical division is the TCP protocol, which is the base for both the FTP- and HTTP protocol. In high-level layers you can have high-level functions e.g. requesting a data transfer process while the low-level layers can perform very specialized and fine-grained functions detailing how the data transfer should be accomplished. One layer thus accomplishes a function by calling several other more fine-grained functions in the layer below. This is illustrated in Figure 2.

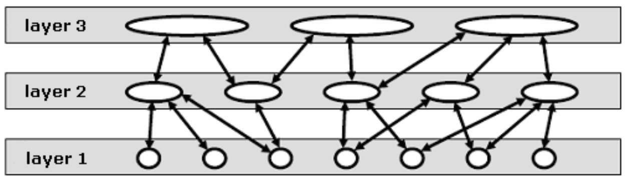


Figure 2 - Vertical division into layers [29]

2.2.3 One-, two-, three and n-tier applications

There are a number of ways a system can be split into physical and logical tiers. How many physical and logical tiers an application should be divided into depends on the use and requirements of the system. In the following we will describe the typical difference between one, two, three and n-tier applications. The following section is based on [16].

One-tier applications

One-tier applications are normally simple isolated programs that don't have network access. Therefore it's not necessary to take network communication and the risk of network failure into consideration. Neither do these programs need to think of synchronization of data, since all the data exists within the same application. When separating tiers physically the application will be slower due to the inevitable fact that communication over network will result in a loss of performance, therefore one-tier applications have high performance.

Two-tier applications

A two-tier application, in contrast to the one-tier application just described, does not combine all functions into a single process but separate different functions. A good example of this kind of application is a chat application. This kind of application contains two separated tiers, a client and a server. The client has the responsibility of capturing user input and displaying the actual messages. The server is responsible of the communication between the people that uses the chat client.

Three-tier applications

A three-tier application, adds another tier to the previous mentioned chat application, this could be in the form of a database. One could think of a three-tier application as a dynamic web application, which has a user interface, business logic and a database each placed in different tiers, this is illustrated in Figure 3. As mentioned in the previous section a two-tier architecture separates the user interface from the business logic, in the same way the three-tier architecture separates the database from the business logic.

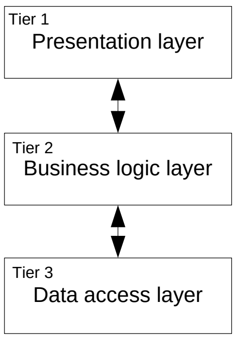


Figure 3 - Three-tier application

Logical n-tier applications

As was mentioned in 2.2.1, logical and physical tiers are not the same. A logical n-tier application is an application where all logical parts are separated into discrete classes. In a typical business application, this generally involves a presentation layer, business logic layer and a data access layer. This was illustrated in the previous section, see Figure 3. This separation also makes the application easier to maintain. The advantages of this architecture are that all business rules are centralized which make them easy to create, use and re-use. The data access is also centralized, which has the same advantage as the centralization of the business rules. Centralizing the data access routines are also good when it comes to maintenance since changes only has to be implemented at one location. The effect of using a middle tier can be seen in Figure 4.

There are really not that many disadvantages of this kind of architecture, however, it takes a bit longer to get up and running since several separate components has to be developed, which also might make the code a bit more complicated to grasp for less experienced programmers. This section was based on [19] and [20].

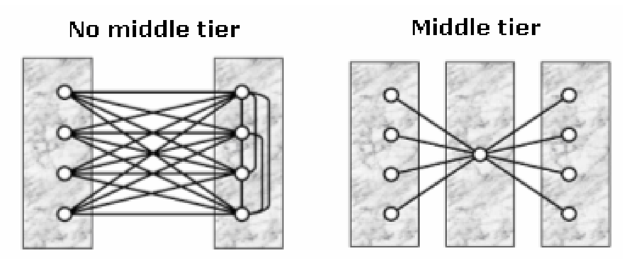


Figure 4 - Middle tier [29]

2.2.4 General advantages of layered applications

There are several advantages of designing applications that are split up into different tiers or layers. Most importantly this form of architecture helps to enforce the principles of high cohesion within system components and low coupling between different components. A system built on these principles will be more robust, easier to adjust, easier to maintain and understand and it allows different software components to be developed in parallel.

The key point is that a system should be split into different smaller parts that are as cohesive and self-governing as possible. Each part has distinct responsibilities and interacts with other parts of the system to accomplish its tasks and responsibilities. This also ensures that the systems can corporate across different platforms and communication protocols and makes it easier to reuse existing solutions to problems often encountered.

All in all these advantages are desirable because they work in the direction of low coupling and high cohesion. The hard part is, however, to implement this in practice and to know when and how it should be implemented. What objects should have responsibility for the different tasks and how do they interact. This is where the design patterns play a vital role.

2.3 Design patterns

A very essential strength of object-oriented software development is the ability to use software classes as more or less independent building blocks that can be reused when appropriate. A pattern is basically a structured description or design of a problem and its solution that can be applied in different situations. If one group of software developers has found a good solution to a problem, there is no reason for another group to reinvent this solution when they encounter a similar problem. Many resources can be saved if it is possible to reuse a design pattern that is known to solve the problem.

It is important to emphasize that patterns are not classes – that is, patterns are not software code, but conceptual designs on how to solve a problem. Patterns can be low-level, concerning a quite detailed solution to a specific problem, or high-level, containing guidelines on how to structure large parts of the system.

Patterns are also a mean of communication between developers. Instead of referring to specific implementation details developers might speak in terms of design patterns.

Throughout this report we will introduce different design patterns that are used in J2EE and .NET. In our sample application we also make use of the Factory Method [9], Façade [9] and Data Transfer Object [8] pattern. In the following we will start by introducing the MVC-pattern in a web context which is closely related to the earlier described logical division between the user interface and the business logic.

2.4 The Model-View-Controller (MVC) in a web context

The following section is based on [18]. The MVC pattern is widely discussed and exists in different contexts. The MVC in a web context is segmented into a Model, a View and a controller. The Model is the gateway to the business logic. It's the model that is responsible for the functionality of the application. The View is the presentation layer. It's the View that provides the user with the available actions and the information needed to use them. The Controller is the bridge between the Model and the View, interpreting requests from the user and commanding the Model and/or View to change as appropriate. The general flow through the MVC is shown in Figure 5.

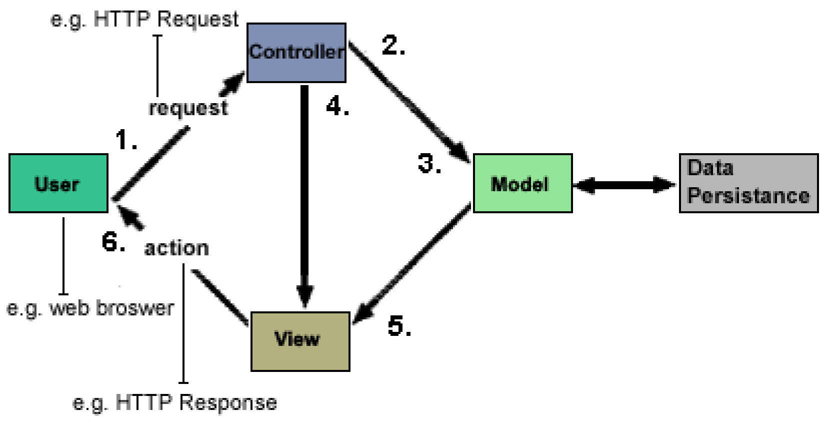


Figure 5 – MVC [18]

In the web context what activates the MVC is when events are generated in the presentation layer. A typically scenario is when a page is requested. Then the MVC handles it in the following way:

- The Event that was generated is handled by the Controller.
- The Controller reads and interprets the event. What actually happens is that the Controller maps the generated Event with the appropriate handler in the Model.
- The Model now performs the action needed. When the Model is finished the result is returned to the Controller.

- Then the Controller determines which View that should be shown. Thereafter everything is forwarded to this responsible View.
- When the responsible View loads it may retrieve data from the Model through a data interface, it may however not execute actions on the Model. The data is now rendered by the View, showing it to the user.

The separation of components created by the MVC pattern helps eliminate UI dependencies, as each component is responsible for a separate set of logically grouped tasks. The separation also reduces code duplication and enhances the maintainability of the application. Further it also makes handling data easier, whether adding new data sources or changing data presentation, because business logic is kept separate from data. Introducing new clients will also be easier since it is not necessary to change the business logic with the addition of each new type of client [14]. Another strength due to the separation of the application into responsibility based parts, is that it's more easy to have different development teams working on what they know best without interfering with each other.

3 Application design in .NET

The purpose of this chapter is to examine how n-tier applications can be designed in .NET. A description of the different possible layered architectures and their advantages and disadvantages will be presented. Next an examination of the features and technologies that might be helpful when implementing the sample application will be described in the layer they belong to. Briefly, an overview of COM+ in .NET will be provided in the business logic layer.

3.1 Layered architecture in .NET

The architectural part in the coming sections is mainly based on [19] and [20].

3.1.1 Two-tier architecture

It's normal to use a two-tiered architecture when programming in ASP.NET using Visual Studio. The reason is that when programming using the drag 'n' drop functionality available a direct data binding is often established between the ADO.NET data and the control the data is bound to.

Direct data binding is not a problem; however it's important to understand when it's appropriate to use it and not to use it. In general direct data binding (two-tier architecture) should not be used when developing to an application where changes and maintenance are important, hence development to an enterprise environment should not in general involve direct data binding.

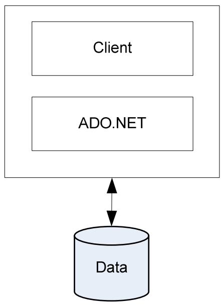


Figure 6 - Two-tier architecture [19]

Advantages

However, direct data binding between ADO.NET and the controls available do have some advantages. Development is extremely fast. Features such as sorting, paging and editing/updating/adding/deleting are functionality that is directly available in the control. It is also easier to get an overview of the code, since all code is gathered in one place.

Disadvantages

The simplicity brought to the application as a result of the direct data binding was mentioned as an advantage in the previous section. However this simplicity does come at a high price, there are several disadvantages.

The direct data binding forces all the business rules to be placed in the front-end of the application. In an application where business rules changes, this demands a lot of maintenance since all clients has to be programmatically updated manually to adopt the new business rules. Hence maintainability is a potential problem, which is generally the case when code duplication exists.

If the data sources changes, e.g. from XML to a database or vice versa, more updating is necessary to be able to load the data into the application. The same dilemma will occur if the database fieldname is changed, then all involved ADO.NET data bindings has to be altered. As described in this section, the disadvantages are mainly due to the lack of a binding gateway, which result in more work when it comes to maintainability since changes needs to be implemented everywhere.

3.1.2 Three-tier architecture

Building a three-tier architecture in ASP.NET could be accomplished in several ways, e.g. through .NET Remoting or through XML Web Services. In the following only the web service version is examined, however, the basic advantages and disadvantages are the same.

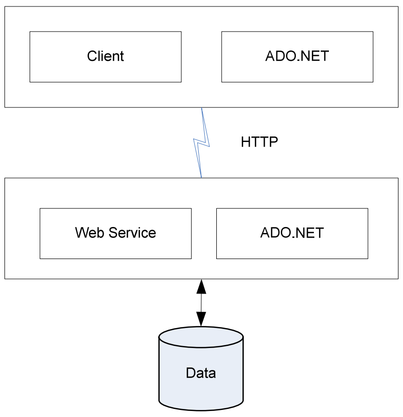


Figure 7 - Three-tier architecture using XML Web Service [19]

Using an XML Web Service

In .NET there are two ways to simply extend the previous described two-tier architecture into a three-tiered one. A possibility is to include a web service to separate the database access into a component. The application would be the exact same as the mentioned in the previous section 3.1.1 however, it would be possible to send and receive data to the application using HTTP. The advantages and disadvantages of this type of three-tier architecture are different from the two-tier architecture just described.

Advantages

The application is now distributed in the sense that users can access it from outside as long as they are connected to the internet. The client maintenance should be minimized since all the ADO.NET connection configuration etc. is placed at the web service. The web service works as a kind of communication gateway which means that all SQL is placed in one location, removing it from the client side and enforcing that the database layer is updated at one place only.

Disadvantages

The business rules are still placed in the front end, as described in the two-tier architecture in section 3.1.1. The extra layer and the HTTP communication make the application as whole work a bit slower. Without internet the application does not function.

3.1.3 Logical n-tier applications

In section 2.2.3 the logical n-tier architecture was examined. To implement this using .NET one would most often create the front-end user interface. This would be done using either WinForms or WebForms (ASP.NET). The business rules would be implemented as a separate Class Library project as would the data layer component. The communication between the client and the data would be done by using web service. It should be mentioned, as can be seen from Figure 8 the business rules are placed on the client side, which minimizes the network traffic. However, this might not be such a good idea if the business rules changes frequently, since an update would be necessary. But .NET is able to copy over new DLLs without registration; therefore this is not a major problem. There are possibilities like having only presentation at the client and the business rules at the server. We will examine this in our sample application.

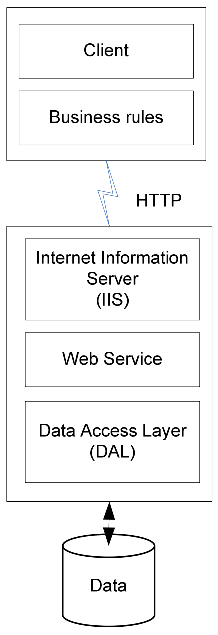


Figure 8 - N-tier application using an XML Web Service [19]

3.2 The presentation layer

This purpose of this section is to examine which possibilities .NET offers to get the data from the data source to the user interface. Further it should be examined what kind of controls there exists to show the data and when to use what control. Due to the wide scope of the project we have decided not to include WinForms but only WebForms in this chapter.

3.2.1 MVC in ASP.NET 2.0

ASP.NET does not implement the MVC in the sense that was described in section 2.4. In ASP.NET MVC is implemented through the use of the "Page Controller pattern" [8]. This pattern, applies the controller at the level of individual pages and not on an application level. If an application level controller is needed, this can be created through the HttpHandler since it intercepts the request before the ASP.NET page does. ASP.NET divides each page into two parts: a View that contains the various controls, and the code-behind, which contains event handlers for every event that the ASP.NET Controller can raise as it processes the page. Developers don't need to interact with the actual control mechanism; they simply write event handlers to wire up the model and view components. What actually happens is that the runtime intercepts the page requests and invokes the requested actions on the model, and determines the correct view to use for the resulting page. All of this is out of site for the programmer. This part was based on [18].

3.2.2 WebForms

WebForms are what ASP.NET web sites are developed around. ASP.NET WebForms have the extension ".aspx" and are the standard way to provide web functionality to clients. A request to an ".aspx" resource will execute the file (the .aspx resource and the code behind file). The code-behind files helps to separate the logic behind a page from the visual part of the page. [6]. This means that the html markup is separated physically from the C# code.

3.2.3 Data binding

The following section is mainly based on [22]. In section 2.2, where the application architecture was discussed it was argued that building a n-tier application involves several layers. Most often a presentation-, business logic- and a data access layer. Communication only takes place between neighbouring layers. Therefore, using the **SqlDataSource** is not always an option, especially not when building n-tiered application, where the user interface communicates with the business logic layer and not directly with the data source.

The SqlDataSource control is a very easy to use and fast to configure. The control refers SQL statements or stored procedures directly from the user interface layer, which is not good design. However, if it's necessary to display database content on a single isolated page, it might be overkill to go through the business logic layer (since this layer, in this case would not serve its purpose). In these situations it might be okay to have the presentation layer (".aspx" pages) call the data access layer directly, hence using the SqlDataSource. Although this is a violation of the guidance provided by the layering pattern, according to [13].

In the case when one is building an n-tier web application it's necessary to have a control that is able to bind to components from the business logic layer. This is exactly what the **ObjectDataSource** does. The ObjectDataSource control enables you to bind controls used in the user interface to render data (e.g. the GridView) to a middle-tier component. The ObjectDataSource is a web site control. In this situation the data is available through the ObjectDataSource. Now a decision about which control to use to show the data has to be taken. There exist about five controls that show the data, each is shortly described in the following. The description is based on [24].

- GridView
- DetailsView
- DataList
- FormView
- Repeater

The **GridView control** displays data as a table and provides the capability to sort columns, page through data, and edit or delete a single record. Due to all the including functionality the control is heavy to use.

The **DetailsView control** renders a single record at a time as a table and provides the capability to page through multiple records, as well as to insert, update, and delete records. A selected record in e.g. a GridView control could determines the record displayed by the DetailsView control.

The **FormView control** renders a single record at a time from a data source and provides the capability to page through multiple records, as well as to insert, update, and delete records, similar to the DetailsView control. However, the difference between the FormView and the DetailsView controls is that the DetailsView control uses a table-based layout where each field of the data record is displayed as a row in the control. In contrast, the FormView control does not specify a pre-defined layout for displaying a record. Instead, you create templates that contain controls to display individual fields from the record. The template contains the formatting, controls, and binding expressions used to lay out the form.

While the **Repeater control** renders a read-only list from a set of records the **DataList control** renders data as table and enables you to display data records in different layouts. The DataList control differs from the Repeater control in that the DataList control explicitly places items in an html table, where as the Repeater control does not.

As has been described in this section different kind of data source controls exists. We described the ObjectDataSource and SqlDataSource. These controls work together with the data-bound controls just described. This is done by binding them to the data source control. When a data-bound control is bound to a data source control, little or no additional code is required for data operations, because the data-bound control can automatically take advantage of the data services provided by the data source control [21]. The overview of what has just been described is shown in Figure 9.

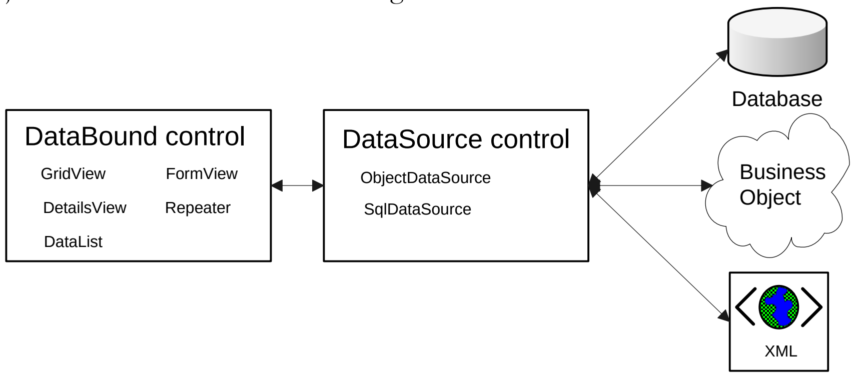


Figure 9 – DataBinding (freely transformed from [21])

When the data is shown in the DataBound control, it is actually presented to the user. Therefore the next section will examine what features ASP.NET 2.0 offers to make the data and the data controls look and feel in a consistent way, throughout a set of web pages.

3.2.4 Master Pages and Themes

ASP.NET Master Pages allow you to create a consistent layout for the pages in your application. A single master page defines the look and feel and standard behaviour that you want for all of the pages (or a group of pages) in your application. You can then create individual content pages that contain the content you want to display. When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page. However, a page does not necessarily have to use the content of the Master Page, each page are able to override the default content of the Master Page showing other content. How Master Pages works is shown in Figure 10. [23]

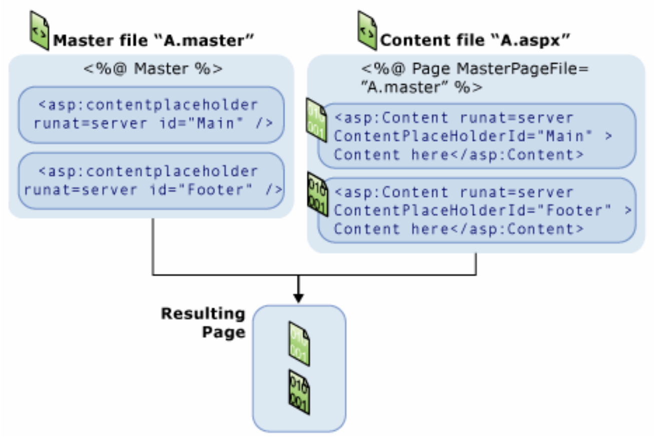

Master file "A.master"
<%@ Master %>
<asp:contentplaceholder runat=server id="Main" />
<asp:contentplaceholder runat=server id="Footer" />
Content file "A.aspx"
<%@ Page MasterPageFile= "A.master" %>
<asp:Content runat=server ContentPlaceHolderId="Main" > Content here</asp:Content>
<asp:Content runat=server ContentPlaceHolderId="Footer" > Content here</asp:Content>
Resulting Page

Figure 10 - Master Pages[23]

Themes enable you to customize server controls with a consistent look. Themes are used together with Master Pages. “A theme is the union of all visual styles for all customizable elements in the pages - a sort of super CSS file”[7]. A theme makes it possible to provide different looking web applications with the exact same source code. A Theme can be programmatically switched to e.g. fulfill a users choice of look and feel on the page.

3.2.5 Sitemap

ASP.NET 2.0 offers a new feature to ease development and maintenance of navigation on web sites. The new control is the **SiteMapDataSource** control. Through this control it’s possible to bind a number of menus e.g. the TreeView control. The control uses a site map provider to read the site map. These nodes are then passed to the TreeView (or another menu) which arranges the HTML rendering. The only problem is that the XmlSiteMapProvider is the only site map provider included in ASP.NET 2.0. The result is that all site maps must be stored in XML files and not in databases or any other persistent media. However it is possible to develop a new provider that reads the information from a database. This was based on [3].

3.3 The business logic layer

In this section we will examine how useful features that are often needed in enterprise applications works in the .NET framework. Features such as thread and transaction management, queued components and security will be examined.

3.3.1 Enterprise Services

Support for Component Services has been extended into the .NET Framework. In this section it's shortly examined how some useful components are used in .NET. This section is based on [13].

It should be mentioned that the functionality of COM+ is exposed in the .NET Framework via the assembly named System.EnterpriseServices, which is part of the Framework class library. Before using it, it has to be referenced from inside the projects. This will make the ServicedComponent class available to all class library projects.

Just-In-Time Activation (JITA)

This feature has the effect that an instance of an object only survives for the span of a single method call, even if a reference to the instance is hold. The main goal of this feature is to optimize the efficiency of stateless components. How this is used can be seen in the code example in the next section.

Object Pooling

An object pool is a number of active instances of a type that COM+ maintains in memory and then dynamically allocates as clients request instances of the type. A pool of objects increase the scalability of an application by avoiding expensive object instantiation and destruction which is more effective than using an instance per client. For this to work, the object must be stateless. Pooling could be used in combination with JITA. This is indicated below:

```
[ObjectPooling(1, 1000)] //Minimum and maximum pool size
[JustInTimeActivation(true)]
public class MyPool : ServicedComponent
{
}
```

Listing 1 - ObjectPooling

If the instance is to be returned to the pool when the method call is completed, `Context.Util.DeactivationOnReturn = true;` should be added before the return statement of the method.

Transactions

COM+ can also manage transactions. Transactions can span methods, components and databases. The problem with COM+ transactions is that they rely on the Distributed Transaction Coordinator (DTC) which consumes a lot of resources. It has some advantages, but if the transactional work does not occur across relational database systems but in a single database, other alternatives might be better. For a single database one could use T-SQL Transactions or ADO.NET connection-based transactions. Transactions in .NET 2.0 are made available in the System.Transaction namespace of the Framework class library. The following method does its work in a transaction:

```
public bool UpdateQuantity(int id, int q)
{
      using (TransactionScope tx = new
             TransactionScope(TransactionScopeOption.Required))
      {
            tx.Complete();
      }
      return true;
}
```

Listing 2 - Transaction

Queued Components

Configuring a component as queued component gives asynchronous method invocation without it being necessary to fix the underlying details of preparing MSMQ messages and placing them in queues. The details of creating a message, putting it in a queue and processing it on the receiver are managed by the hosting environment, hence the Microsoft Transaction Server. When a component is configured as queued, a call to the message prompts COM+ to prepare a MSMQ message and place it in a private queue. Another COM+ process acts as a listener to this queue, pulls the message out when it arrives and invokes the methods described in the message. It is beyond the scope of this project to examine queued components any further.

Role-Based Security

COM+ enables roles at the component, interface or method level. Role creation is done with an assembly-level attribute. The example in Listing 3 would force that a caller to this class are in the Professor role.

```
[assembly: SecurityRole("Professor")]
```

This would result in the corresponding COM+ roles that are created. Now the SecurityRole attribute can be used e.g.:

```
[SecurityRole("Professor")]
public class Student
{
}
```

Listing 3 – Security

This section has briefly provided information about how complex features could be used in .NET at the business logic layer level. Since the business logic layer communicates with the data access logic layer, next section will describe this layer.

3.4 The data access layer

When building an application it has to be considered how the database communication is handled. The mapping could primarily be done in two different ways, either through an Object/Relational (O/R) mapping tool such as NHibernate or manually. In the following section a discussion about the data layer in .NET follows. This section focuses on ADO.NET, not on using an O/R mapping tool.

3.4.1 ADO.NET 2.0

"ADO.NET is the primary relational data access model for Microsoft .NET-based applications. It may be used to access data sources for which there is a specific .NET Provider, or, via a .NET Bridge Provider, for which there is a specific OLE DB Provider, ODBC Driver, or JDBC Driver. ADO.NET is sometimes considered an evolution of ADO technology, but it is important to note that some major changes were made between the two."[34]. The rest of this section is primarily based on [15].

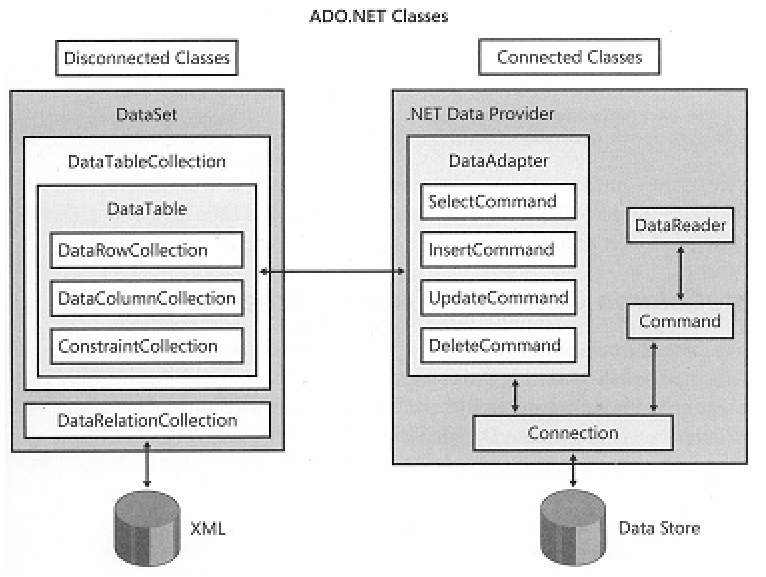


Figure 11 - ADO.NET classes [10]

Minimized Open Connections

ADO.NET minimizes the time with open database connections since a connection is only established to retrieve and update records. The records received are copied into a DataSet where after the connection is closed. The DataSets are designed to store data in disconnected and distributed data environments [15]. The ADO.NET classes can be seen above in Figure 11.

.NET Data Providers

The .NET data providers enable an application to connect to a data source, execute commands and receive results. The .NET Data Provider consists of four components. These components with explanation are as follows:

- Connection
- Command
- DataReader
- DataAdapter

The **Connection** object enables you to establish the connection to the data source. The **Command** object enables you to execute SQL commands (Read, Insert, Update, and Delete) and retrieve results from a database. The **DataReaders** provide forward-only, read-only, connected access to data sources, no natively sorting functionality and they don't support data manipulation however they are very fast. In remoting there is a risk that the client application holds the database connection open for lengthy periods. This is the case since once a DataReader has been opened, the connection it is using cannot be used by any other object and the connection to the data source is maintained until the close method is called. The **DataAdapter** object enables a database and a DataSet to communicate with each other; hence you use the adapter to transfer data between a DataSet and a data source.

DataSets

This section about DataSets is based on [1]. A DataSet is an in memory database specialized to hold any kind for tabular data. DataSet allow relations to be defined between pairs of tables. DataSets are designed to handle data from different sources, such as the file system, memory, a database etc. DataSets have native support for XML serialization and optimistic concurrency. DataSets are also very easy to use when working with either Web Forms or Windows Forms. The previous described DataAdapter is used to bridge the DataSet with the data source. By calling methods on a data adapter, you can execute database commands using the contents of the DataSet.

As was mentioned before, DataSets contains tabular data, they do however not ad any specific additional behaviour. In situations where objects need behaviour datasets might not be suitable. The fact that DataSets are not business entities but merely holds data in tabular form makes the programming more relational than object oriented.

Strongly Typed DataSets

The difference between a typed DataSet and a normal DataSet is that the typed version provides strongly typed methods, events and properties. The typed DataSet derives from a DataSet and therefore inherits all methods, events and properties of a DataSet. Beside that the strongly typed DataSet gives type safety of values at compile time. It also provides IntelliSense in Visual Studio and making the work simple due to the fact that tables and columns are accessed by name. It makes the database programming more object-oriented-like.

Creating a strongly typed dataset in Visual Studio .NET is easy, basically the only thing that has to be done is to drag a database table from the Server Explorer onto a DataSet designer (XSD file). This XSD file then stores the XML that defines the schema for the strongly typed DataSet. This section was based on [2].

3.4.2 Data Access Application Block

The patterns & practices Enterprise Library is a library of application blocks designed to assist developers with common enterprise development challenges. Microsoft has developed a Data Access Application Block which simplifies development tasks that implement common data access functionality. Applications can use this application block in a variety of situations, such as reading data for display, passing data through application layers, and submitting changed data back to the database system. [32]

3.4.3 Handling concurrency

In principle concurrency can be handled either pessimistic or optimistic. If a user reads data with the intention of updating it, under pessimistic concurrency a lock is established on the data source (row) during the whole operation. The data source (row) is first available after the operation. When using optimistic concurrency, a user does not lock the data source (row) when reading it, however it has to be determined before the update takes place whether the data source (row) has been changed since it was read. In the following we explain how optimistic concurrency can be implemented. This section is based on [5].

Distributed time stamps

One solution is to add a timestamp column to each data row. If the timestamp is returned with each query, it's fairly simple to verify whether the time stamp value in the database is the same as the original time stamp, before updating. Optimistic concurrency violation would occur if the timestamps are not the same.

Maintain original values

Another solution is to maintain a copy of the original values. Before updating it's only necessary to verify that the original values match with the ones stored in the database. It should also be mentioned that maintaining the original values gives the possibility to check each value instead of the whole row, which is the case when using distributed time stamps.

DataSets

DataSets has native support for optimistic concurrency. This is because the DataSets maintain a copy of the original values.

3.5 New features in ASP.NET 2.0

Throughout this section we have highlighted some of the new features of ASP.NET 2.0 that we intend to use in the sample application. We have talked about Master Pages, Data Source Controls, Data Controls and Data-Driven Site Navigation. We have also indirectly talked about Codebehind 2.0 and some of the other new controls. However there are still quite a few new features that due to the scope of this project will not be examined. These features can be seen in Appendix 1.

4 Sample application goals

In order to fulfill the goals of the project, namely to get familiarized with C#, ASP.NET 2.0, Visual Studio and to investigate how to design and implement a layered architecture, we will develop a small sample application. The sample application is based on a general business case, namely a company that sells products through the internet. Therefore the company has an existing database containing customers, orders, products etc.

The sample application should allow to select a customer and to view all the orders for that given customer. For each order it should be possible to see the orderlines and the actual product. It should be possible to view the above mentioned information as a WebForm and WinForm.

Since this project focuses on the architecture and that the time available for the project is limited, the sample application chosen is very simple and no energy will be placed on the graphical layout.

4.1 The architecture

As was mentioned in our problem definition the purpose of this project is partly to challenge the typical 2-tier application design that is encouraged in .NET. Therefore our implementation of the sample application should use the previous explained n-tier architecture in .NET. In our case this should be a logical three-tier architecture with a presentation layer, business logic layer and data access layer.

Our application is simple; however, it's not difficult to imagine scenarios where this type of application needs "Enterprise Services" as described previous. Therefore the architecture should also enable these to be easily configured.

This architecture should provide us with a middle tier; the business logic layer that makes the presentation layer independent of the data access layer. We should furthermore ensure loose coupling between the business logic layer and the data access layer in order to be able to change aspects of the system without too much affect on the other.

The presentation layer should use the right controls and navigation functionality provided by ASP.NET 2.0.

The implementation of the sample application should illustrate blueprints of a .NET-based layered application architecture.

4.2 The Database

The database used to illustrate this company is the Northwind sample database from Microsoft. The database design is available in Appendix 2. Since we previously discussed strongly typed DataSets, we have provided the "database diagram" as a DataSet (.xsd file), which could be used for programming the sample application using strongly typed DataSets.

5 Implementation of sample application

Now that we have examined how .NET-based layered applications could be designed and have established goals for the sample application we will present our implementation. As described in the problem definition, our intention was to challenge the typical two-tier architecture of ASP.NET. However, as it was described in section 3.1.1 there are scenarios where two-tier architecture is preferred due to the scope and requirements of the application. In relation to this it should be said that our architecture might seem a bit over-engineered for this simple application. But it provides the flexibility to plug in new features and establishes a loose coupling between the different layers. Inspiration for our implementation has been found in [30] and [31].

In the following we will start by introducing the overall architecture of the application and how the different layers is interconnected and communicates. After that a more detailed description of the system in its whole and the contents of each tier follow.

5.1 Architecture

We have designed the sample application as a logical three-tier architecture with a presentation layer (UI), business logic layer (BLL) and data access layer (DAL). The UI is implemented both as a WinForm and as an WebForm (ASP.NET 2.0) application to show that the architecture is able to service different clients. Because the WinForm is placed on a different physical tier than the rest of the application, the BLL is exposed to the UIs through an XML Web Service. The sample application architecture is shown in Figure 12.

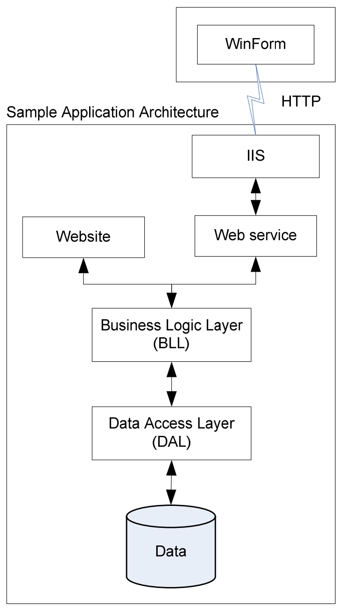


Figure 12 - Sample application architecture

The website can either use this web service or access the BLL component directly. Since the website is hosted in the same environment it is unnecessary to go through the web service so the BLL component is accessed directly. However, the web service makes it possible to host the website on another physical tier than the BLL and DAL. Beside for the logical separation into tiers we compiled different parts into C# class libraries. The whole solution can be seen in Figure 13.

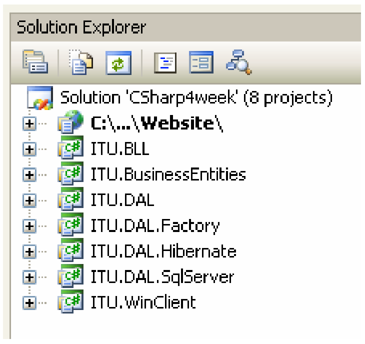


Figure 13 - Sample application Solution Explorer

For instance the BLL is implemented as a C# class library that is compiled into one assembly. It is the middle tier and acts as a Façade[9] to the rest of the system. It provides the methods to handle creation and retrieval of customers, orders etc. The methods in the BLL doesn't contain that much "business logic" but with this design it is possible to implement more intelligent methods that for instance perform different enterprise services such as distributed transactions. However, we haven't implemented them due to the time limit of this project. Enterprise Services was discussed and small implementation examples where provided in section 3.3.1.

The BLL retrieves persistent data only through the DAL. The DAL encapsulates all communication with the database and offers different data providers. We have implemented a Microsoft SQL Server database and a NHibernate data provider.

To pass information between the tiers we have created Business Entities, also known as a "Data Transfer Object" [8]. This enables the use of more course-grained operations. The Business Entities such as Customer is light-weight objects that simply represent data from the database. But since it is plain C# objects we can return a Collection of the objects and send them all the way to the UI that can print them in different DataBound controls, e.g. a GridView.

5.2 System overview

To give a better overview of how the application is structured and how the different tiers / layers communicate we have provided a class diagram of the main aspects of the application that can be seen in Figure 14. This is a simplified diagram that only deals with customers. There is for instance only an ICustomerDataProvider interface in the DAL and a CustomerManager class in the BLL. In the implementation we also have classes and interfaces to deal with Orders, OrderDetails and Products.

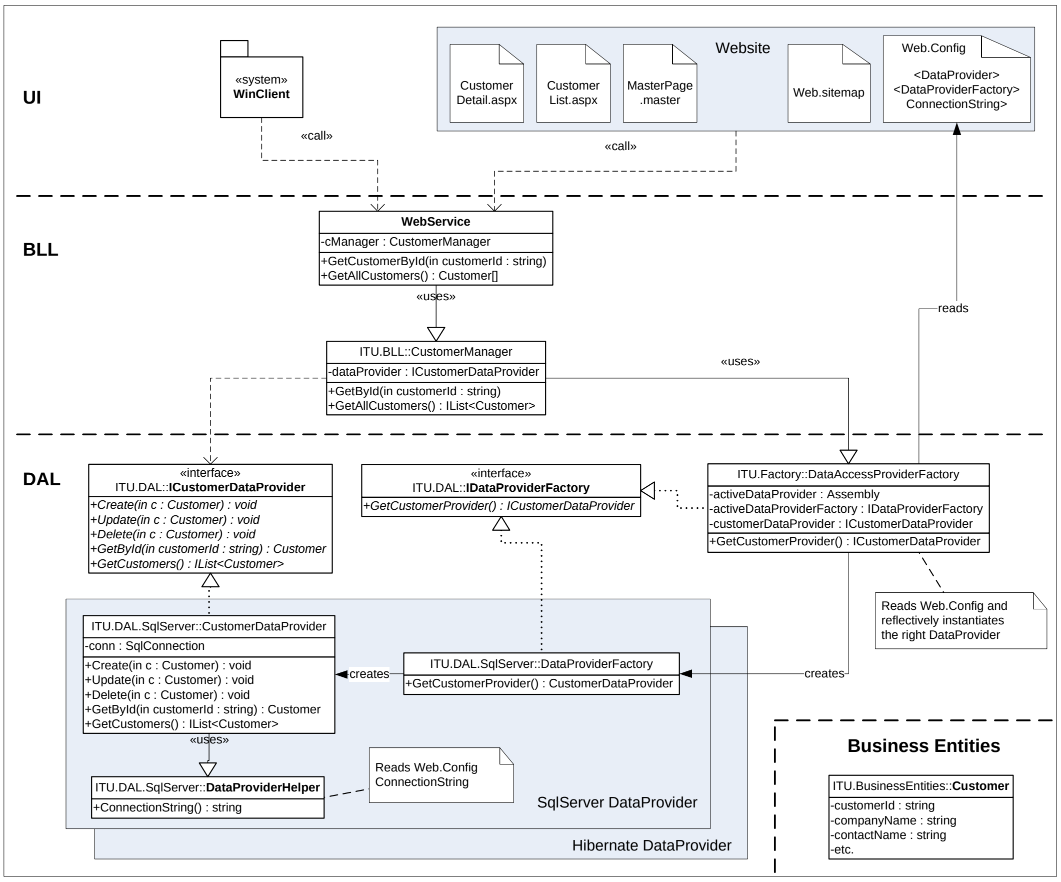

UI
«system» WinClient
Website
Customer Detail.aspx
Customer List.aspx
MasterPage .master
Web.sitemap
Web.Config
<DataProvider>
<DataProviderFactory>
ConnectionString>
«call»
«call»
BLL
WebService
-cManager : CustomerManager
+GetCustomerById(in customerId : string)
+GetAllCustomers() : Customer[]
«uses»
reads
ITU.BLL::CustomerManager
-dataProvider : ICustomerDataProvider
+GetById(in customerId : string)
+GetAllCustomers() : IList<Customer>
«uses»
DAL
«interface» ITU.DAL::ICustomerDataProvider
+Create(in c : Customer) : void
+Update(in c : Customer) : void
+Delete(in c : Customer) : void
+GetById(in customerId : string) : Customer
+GetCustomers() : IList<Customer>
«interface» ITU.DAL::IDataProviderFactory
+GetCustomerProvider() : ICustomerDataProvider
ITU.Factory::DataAccessProviderFactory
-activeDataProvider : Assembly
-activeDataProviderFactory : IDataProviderFactory
-customerDataProvider : ICustomerDataProvider
+GetCustomerProvider() : ICustomerDataProvider
Reads Web.Config and reflectively instantiates the right DataProvider
ITU.DAL.SqlServer::CustomerDataProvider
-conn : SqlConnection
+Create(in c : Customer) : void
+Update(in c : Customer) : void
+Delete(in c : Customer) : void
+GetById(in customerId : string) : Customer
+GetCustomers() : IList<Customer>
creates
ITU.DAL.SqlServer::DataProviderFactory
+GetCustomerProvider() : CustomerDataProvider
creates
«uses»
ITU.DAL.SqlServer::DataProviderHelper
+ConnectionString() : string
Reads Web.Config ConnectionString
SqlServer DataProvider
Hibernate DataProvider
Business Entities
ITU.BusinessEntities::Customer
-customerId : string
-companyName : string
-contactName : string
-etc.

Figure 14 - Class diagram dealing with Customers

5.3 Business logic- & data access layer

To ensure loose coupling between the BLL and the DAL we use Factory Methods [9] that reads the Web.Config configuration file and reflectively instantiates objects that the BLL uses to communicate with the DAL. In this way by changing an entry in the Web.Config the application will use another data provider and therefore even another database. This is done without recompiling anything. The sample code in Listing 4 is from our Web.Config file. The ITU.DAL.Hibernate version of the data provider is out-commented; however, if the other data provider is out-commented instead, the Hibernate version would be used at runtime.

```
<!--
<add key="DataProvider" value="ITU.DAL.Hibernate"/>
<add key="DataProviderFactory" value="ITU.DAL.Hibernate.DataProviderFactory"/>
-->

<add key="DataProvider" value="ITU.DAL.SqlServer"/>
<add key="DataProviderFactory" value="ITU.DAL.SqlServer.DataProviderFactory"/>
```

Listing 4 - Web.Config

Furthermore we use the pattern Separated Interface [8] and are always coding against an interface and not the real implementation. In this way we can easily provide a new implementation of the DAL called a data provider that supports another kind of database and change the configuration file to use the new implementation.

The DAL defines interfaces to work on e.g. customers. The ICustomerDataProvider interface defines the typical CRUD functionality for working on that a data provider has to implement. As mentioned we have created two different data providers / implementations that work on SqlServer and Hibernate respectively.

The DAL also defines an IDataProviderFactory interface that data providers need to implement. The implementation of this interface is used within the data provider to create the right object of the type ICustomerDataProvider. You could imagine that each data provider have different objects that could be used, e.g. one that used optimistic concurrency and one that used another strategy. The ICustomerDataProvider interface that defines the CRUD methods is shown in Listing 5.

```
namespace ITU.DAL
{
      public interface ICustomerDataProvider
      {
            void Create(Customer c);
            void Update(Customer c);
            void Delete(Customer c);

            Customer GetById(string customerId);

            DataSet GetCustomersDataSet();
            void UpdateCustomersDataSet(DataSet ds);

            IList<Customer> GetCustomers();
      }
}
```

Listing 5 - ICustomerDataProvider

5.3.1 Communication through the tiers

In the following example we assume that the Web.Config file is set up to use our SqlServer data provider.

When either the client running the WinForm or the website requests the web service for a list of customers, the web service calls the GetAllCustomers() method on the CustomerManager. The CustomerManager then uses the DataAccessProviderFactory to get an object that implements ICustomerDataProvider (here SqlServer.CustomerDataProvider) on which the GetAllCustomers() method can be called, which return the actual list of customers as a IList of ITU.BusinessEntities.Customer. To do this the DataAccessProviderFactory first reads the Web.Config file and reflectively creates an object that implements IDataProviderFactory (here SqlServer.DataProviderFactory). This factory object creates and returns the right CustomerDataProvider object on which the CustomerManager can invoke the GetAllCustomers() method and return the list of customers to the web service which returns it to the client.

The following code shows what happens in the constructor of the DataAccessProviderFactory. The providerName and providerFactoryName is read from the Web.Config file whereafter the assembly is loaded and an instance of the right DataProviderFactory is created.

```
string providerName = ConfigurationSettings.AppSettings["DataProvider"];
string providerFactoryName =
ConfigurationSettings.AppSettings["DataProviderFactory"];
activeDataProvider = Assembly.Load(providerName);
activeDataProviderFactory =
(IDataProviderFactory)activeDataProvider.CreateInstance(providerFactoryName);
```

Listing 6 - Loading the DataProviderFactory

5.3.2 The implemented data providers

In the following we will shortly describe features of the two implemented data providers. As was mentioned earlier the different data providers implement the different interfaces in the DAL.

SqlServer data provider

In the SqlServer data provider all methods etc. needed in the application has been implemented against Microsoft SQL Server database. Our application has not been designed to work on DataSets but on objects, therefore the primary focus of this layer is to load Business Entities with data from the database. Since we do not use DataSets which has native support for concurrency, we have to implement this manually. However the sample application is built on top of Microsofts Northwind sample database and this database does not use Timestamp in the database columns. To be able to handle concurrency we could have modified the database or we could have used the other approach described in 3.4.3, hence keeping a copy of the original values. Then the concurrency would have been implemented by verifying that the values haven't been changed from the read operation. If the read operation looked like, `SELECT Column1, Column2 FROM Table1`. Then the update would look like:

```
UPDATE Table1
Set   Column1 = @NewValueColumn1,
      Set     Column2 = @NewValueColumn2
      WHERE   Column1 = @OldValueColumn1
      AND     Column2 = @OldValueColumn2
```

Listing 7 – Concurrency

If a concurrency violation occurs an exception could be thrown and handled. We have not implemented this.

It was just mentioned that the application was not designed to work on DataSets, however we have provided a couple of methods that actually works with DataSets, to be able to "feel" the difference and learn how DataSets work. Our data access is primary done using SqlCommand and SqlDataReader. An example of how this is implemented can be seen in Listing 8.

```
public IList<Product> GetProducts(string[] productId)
{
      IList<Product> products = new List<Product>();
      try
      {
            string query = String.Format("SELECT ProductID, ProductName,
            SupplierID, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock,
            UnitsOnOrder, ReorderLevel, Discontinued FROM products WHERE ProductID
            IN ('{0}')", productId);

            SqlCommand myCommand = new SqlCommand(query, myConnection);
            myConnection.Open();
            SqlDataReader myReader;
            myReader = myCommand.ExecuteReader();
            while (myReader.Read())
            {
                  //Add the product to the collection
            }
      }
      finally
      {
            myReader.Close();
            myConnection.Close();
      }
      return products;
}
```

Listing 8 – GetProducts

As can be seen from the code in Listing 8 the SQL statement is coded within quotations. There is no compile time check of the SQL statements which is dangerous due to the potential of runtime errors. If we would have used strongly typed DataSets, there would have been IntelliSense and compile time check of the SQL provided by the DataAdapter.

Hibernate data provider

In the Hibernate data provider we use the ORM persistence framework NHibernate. NHibernate is a .NET port of the popular Hibernate framework for Java. In NHibernate one are persisting C# classes (in our case the business entities such as Customer) to a database via an XML mapping file. In this mapping file you specify how the fields of an object are going to be mapped to the relational database. NHibernate supports mapping of complex object structures which uses inheritance and collections of other objects as fields. The advantage of using NHibernate is that the framework handles all communication to and from the database and constructs the SQL automatically for the target database. An example of a mapping file from our sample application can be seen below in Listing 9.

```
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
      <class name="ITU.BusinessEntities.Customer, ITU.BusinessEntities"
             table="Customers">
            <id name="CustomerID" column="CustomerId" type="String(5)">
                  <generator class="assigned" />
            </id>
            <property name="ContactName" column="ContactName"
                      type="String(30)" not-null="false" />
            <!-- More properties follows here -->
      </class>
</hibernate-mapping>
```

Listing 9 - Customers.hbm.xml

In the mapping file we are specifying that the Customer class in the assembly ITU.BusinessEntities is going to be mapped to the Customers table in the database. Furthermore we specify the primary key and some properties of the Customer object.
This mapping file is read by the NHibernate framework when we create a new Configuration and ISessionFactory as in the example code in Listing 10.

```
internal class HibernateHelper
{
      private static readonly Configuration config;
      private static readonly ISessionFactory sessionFactory;

      static HibernateHelper()
      {
            config = new Configuration();
            config.AddAssembly("ITU.DAL.Hibernate");
            sessionFactory = config.BuildSessionFactory();
      }
      // Get methods for config and sessionFactory should be here
}
```

Listing 10 – HibernateHelper.cs

When we have an object of ISessionFactory we can create a new ISession on which we can open a session.

```
public abstract class CommonDataProvider
{
      protected ISession GetNewSession()
      {
            return HibernateHelper.GetSessionFactory().OpenSession();
      }
      protected void EndSession(ISession session)
      {
            session.Flush();
            session.Close();
      }
}
```

Listing 11 – CommonDataProvider.cs

The OpenSession() method opens a connection to the database that will be used to persist the data in an object. We use the ISession returned from GetNewSession() in our code to persist a Customer object by calling the Save() method as shown below.

```
public class CustomerDataProvider : CommonDataProvider, ICustomerDataProvider
{
      public void Create(Customer c)
      {
            ISession session = GetNewSession();
            session.Save(c);
            EndSession(session);
      }
      // More CRUD methods here
}
```

Listing 12 – CustomerDataProvider.cs

The above example is very simple but NHibernate provides lots of possibilities for advanced O-R mapping and querying against the data. Due to the scope of the project we will not go into further details about NHibernate. For more code samples we refer to the source code in Appendix 4.

5.4 Presentation layer

The implementation provides both a WebForm (ASP.NET) and a WinForm application. In this section our primary focus is to describe the implementation of the WebForms.

5.4.1 ASP.NET 2.0 website

Previous it was described that the layering pattern only allows the user interface to communicate with its neighbouring layer. Since we have a business logic layer that uses business entities to transport the data through the layers, the user interface make use of the ObjectDataSource as a DataSource Control. The ObjectDataSource and other controls were described in section 3.2.3. In Figure 15 it is shown that the ObjectDataSource is configured to receive data from the web service.

It's important to make sure that one does not try to use any data type that is not supported by the XML Web Service using the SOAP protocol. As mentioned we use business entities to pass the data through the tiers. No problems should be encountered since XML Web Services using the SOAP protocol support "Class and struct types with public fields or properties. The public properties and fields are serialized. Classes must have a default constructor that does not accept any parameters" [27].

From Figure 15 it can be seen that the GridView is configured to use the ObjectDataSource as its data source. The GridView is placed in the WebForm, hence the .aspx page. The Custom content placeholder does not default to the content of the Master Page, since we wish to display the data configured in the ObjectDataSource, and not any content from the Master Page. However, the other content placeholder does default to the content of the Master page. From Figure 15, it can be deducted that the left content placeholder implements a TreeView, which is connected to a SiteMapDataSource, which reads its content from a SiteMap file. The SiteMap file is an XML file where we have defined the navigation we wish to use in the application.

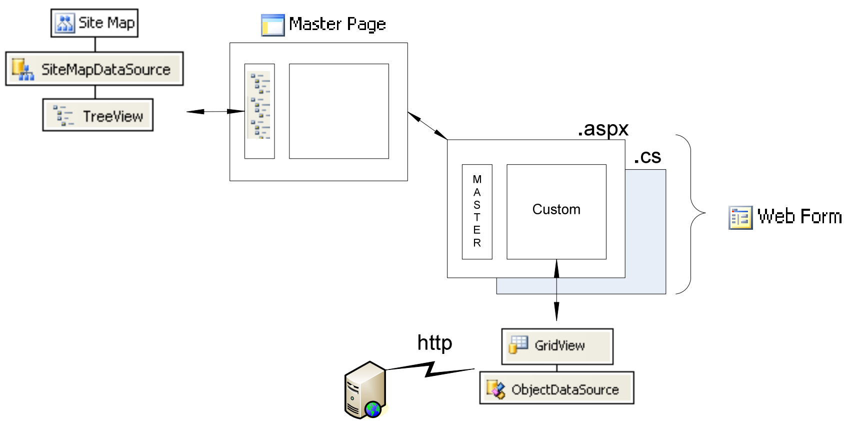


Figure 15 - Sample Application UI website solution (own creation)

We have implemented a couple of WebForms to be able to handle the sample application. We have a CustomerDetail.aspx where the details for a given customer is shown and possibly updated. Before the customer details are shown the customer is chosen from a list. CustomerList.aspx shows all the customers, and enables updating and a link to all the orders for each customer. The CustomerOrders.aspx shows all the orders for a given customer, again each order consist of a number of orderlines, these are shown by the OrderDetails.aspx. Screenshots of the CustomerList WebForm is shown in Appendix 3.

5.4.2 WinClient

The WinForm does not operate and look in exactly the same way as the WebForm. The WinForm providing the same services, calling them from the same web service looks as in Appendix 3.

Using asynchronous calls from WinClient

A normal, or synchronous, page holds onto the thread for the duration of the request, preventing the thread from being used to process other requests. If a synchronous request becomes I/O bound—for example, if it calls out to a remote web service or queries a remote database and waits for the call to come back—then the thread assigned to the request is stuck doing nothing until the call returns [4]. As just described, IO-related tasks should be considered to be performed asynchronously. Since web service calls can take a long time to return, pages that execute them are ideal candidates for asynchronous processing. We have implemented both synchronous and asynchronous calls to the web service in the WinClient. The asynchronous calls are done in the following way:

```
/*
 * Async call to webservice, get all the customers in a collection
 * This will only work in ASP.NET 2.0
 */
private void btn_CollectionCustomerAsync_Click(object sender, EventArgs e)
{
    wS.GetAllCustomersCompleted += new
localhost.GetAllCustomersCompletedEventHandler(GetAllCustomersCompleted);
    wS.GetAllCustomersAsync();
}

void GetAllCustomersCompleted(object sender,
localhost.GetAllCustomersCompletedEventArgs e)
{
    localhost.Customer[] customers = e.Result;
    this.UpdateView(customers);
}
```

Listing 13 - Asynchronous calls

The above example is new in the .NET Framework 2.0 and is only supported by the .NET Framework version 2.0 web service proxy. The new service proxy includes a method named *NameOfMethod*Async and an event named *NameOfMethod*Completed. You can call *NameOfMethod* asynchronously by registering a handler for *NameOfMethod*Completed events and calling *NameOfMethod*Async. This was done in Listing 13. The above code is based on [4].

Another thing that was a bit different between the WinForm and the WebForm was the DataBound and DataSource controls. In the WinForm we use BindingList<T> when displaying the list of all customers. This enables the DataGridView (this control is not available in ASP.NET 2.0 websites) to a list of Customers where Customer is a class with properties like ID, Name, etc. The code in Listing 14 solves the problem:

```
private void UpdateView(localhost.Customer[] customers)
{
    BindingSource bs = new BindingSource();
    BindingList<localhost.Customer> cList = new BindingList<localhost.Customer>();

    foreach (localhost.Customer c in customers)
    {
        cList.Add(c);
    }

    bs.DataSource = cList;
    dataGridView1.DataSource = bs;
}
```

Listing 14 – BindingList (code based on [28])

"dataGridView1" will show all the Customers in the BindingList and display columns for each of the properties on Customer i.e. it will have columns for ID, Name, etc. This is shown as the upper-most left window in the WinForm screenshot in Appendix 3. The advantages of BindingList<T> are that as you add, remove, insert Customers into this BindingList the DataGridView will show up the changes and correspondingly add, remove and insert rows [28]. The same approach was used for each DataGridView in the WinForm.

5.5 Further implementation issues

In addition to the above description of our implementation it should be mentioned that our code certainly lack certain things. As an example we have not applied any error handling strategy such as defining customer exceptions for various situations. One of the main objectives of good programming is to create robust programs. To be able to do this a good error handling strategy is a must since it will minimize the probability that the program will crash. However, our code should not be viewed as "ready to ship" but be seen as a guideline for how an application could be designed and for us to get familiar with the C# and ASP.NET languages.

There are certain things that must be implemented before any of our code could ever be considered for production, this is mainly unit testing, exception handling and concurrency. In addition to exception handling it should also be mentioned that when working with web services exceptions thrown are sent back to the client in the form of a SOAP fault. Therefore it's necessary to investigate further on handling and throwing exceptions in XML Web Services and displaying it presentable to the user.

6 Comparison of J2EE and .NET

At this point we have examined layered application architecture both in general and in .NET. Furthermore we are now familiar with some of the technologies in .NET in the different layers / tiers. It would be interesting to investigate how all this relate to the J2EE world. The purpose of the following sections is to compare the J2EE and .NET technologies primary in the scope of this project, namely application architecture. However, we will not only compare on what we previously has examined in the .NET environment but also to give a complete overview, a top-to-bottom comparison of the two platforms. This chapter is primarily based on [18] and [12].

6.1 Overview of the platforms

The complete overview of the similarities and differences between the two technologies and how they relate to one another are summarized in Figure 16. The figure is based on [18] but has been modified to be up-to-date with the most recent changes in the technologies.

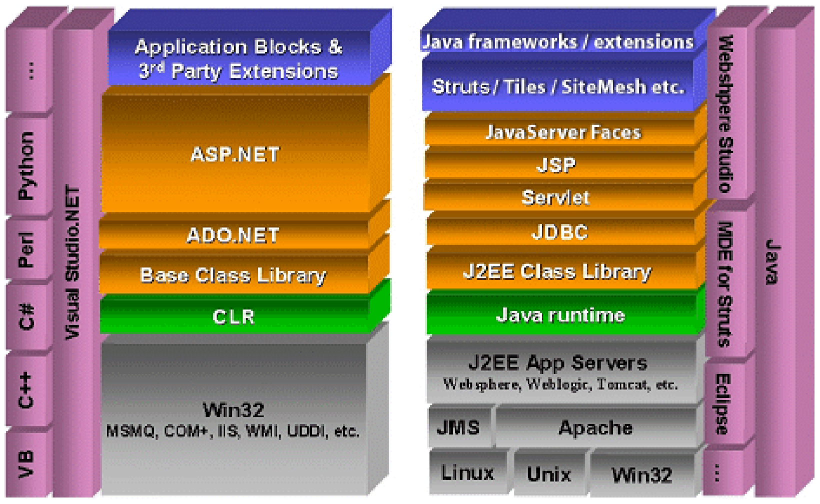


Figure 16 - NET and J2EE platform stack (freely transformed from [18])

In the remainder of this section we will focus on the similarities and differences in Figure 16. The purpose is not to give a complete elaboration, but to present a small summary about the two technologies.

6.2 Language and IDE

One of the major differences between the two platforms is the language support. As Figure 16 shows, J2EE only has support for the Java language, where .NET has support for multiple languages. For a complete list of language support see [25].

With regards to the integrated development environment (IDE) J2EE has a broad selection to choose from. Examples are the commercial IBM WebSphere Studio Application Developer, BEA WebLogic Workshop or Eclipse as the open-source alternative.

In .NET there is one primary choice of IDE, namely Visual Studio that supports development of all the different kind of application in the .NET framework.

6.3 Application servers and operating systems

Both J2EE and .NET rely on one or more servers to host their applications and to provide essential services such as transactions, security, logging, messaging etc. These servers, and the runtimes themselves, also rely on the base operating system.

J2EE is designed for platform independence and many vendors have built J2EE application servers that run on a variety of operating systems. As such J2EE is very flexible but the platform independence has associated costs such as loss in performance because of missing native integration with the OS.

Unlike J2EE the .NET application server is the Windows Operating System and the services it provides. .NET is closely integrated with IIS and COM+ to provide reliable application hosting services. Therefore .NET runs in a closely integrated environment but also need to run on that exact platform. However, various projects are working on porting the .NET framework to other platforms (e.g. the Mono project for Linux, Unix etc.).

6.4 Runtime engines

Both J2EE and .NET run on top of a runtime engine that interprets and runs the code and provides critical services such as memory management and garbage collection. Source files are compiled into bytecode or intermediate language that is interpreted into native machine language at runtime. However there is one big difference between the two. Java is designed for platform independency where the .NET Common Language Runtime (CLR) is designed for optimized performance on the Windows operating system.

6.5 Class libraries

Both J2EE (or just Java) and the .NET framework provides a big class library with lots of classes. In Java it is often just called the Java API while in .NET it is called the base class library (BCL). In Java the API is organized into packages, which are logical groupings of the classes while it is called namespaces in .NET.

6.6 Technologies in the different layers

As this project is concerned about application design of layered architectures we will in the following sections compare the technologies in the presentation layer, business logic layer and data access layer in J2EE and .NET.

6.6.1 Presentation layer

Both J2EE and .NET offers various web application technologies. In .NET web applications are built using ASP.NET WebForms as described earlier and include different UI controls such as buttons, text fields, GridViews etc. Furthermore ASP.NET is an implementation of the MVC design pattern and the programming model is event-driven as opposed to the old ASP model. ASP.NET provides many valuable features also described earlier.

In J2EE the basic technologies to build web applications is Servlets and JavaServer Pages (JSP). These correspond roughly to the old ASP model, where html markup and scripting is mixed together in one file. Because of problems with this kind of development different Java web frameworks has emerged such as Struts, Tapestry and JavaServer Faces (JSF). All of them implement the MVC pattern to ease the development of web applications. The latter (JSF) is furthermore an UI based framework based on components comparable to ASP.NETs controls and offers many of the same features. JSF is a specification from Sun as is the rest of the J2EE framework. This enables different vendors to provide an implementation of JSF and makes it competitive for the vendors to provide something special and to extent the framework.

ASP.NET vs. JSF

ASP.NET and JSF share many similarities but also differ on some points. Since we think that the presentation layer is quite appealing and ASP.NET and JSF is interesting technologies we will make a somewhat detailed comparison of their features and architecture.

Implementation of MVC

There is a big difference between the two in their implementation of the MVC pattern. The JSF architecture is implemented using a Front Controller pattern [8] called the FacesServlet. This Servlet is responsible for all incoming requests and delegates control to the different classes that take care of validation, navigation, updating model values etc. This means that the Controller is application-wide and every request goes through the FacesServlet in opposite of ASP.NET which uses the Page Controller pattern as described in section 3.2.1.

Backing beans / code-behind files

Both ASP.NET and JSF make it possible to separate UI markup such as html from the actual code in C# or Java. In ASP.NET this technique is called code-behind files and these classes define the event handlers and other code belonging to the page. From these event handler methods you can access the UI components programmatically, change properties etc.

JSF uses the notion of "backing beans" to separate code from markup. Backing beans are JavaBeans that contain action and actionListener methods as well as properties that can be bound to UI components such as input fields and buttons. In this way you can programmatically access the components in the UI as with ASP.NET and respond to the user's actions like clicking buttons and updating input fields. This actually leads to the event-based programming model that both ASP.NET and JSF provides.

Event-based programming

Both ASP.NET and JSF delivers the intuitive event-based programming which makes development of web applications more like the old Visual Basic UI development and the current Windows Forms environment. However, the way it is implemented differs.

In ASP.NET any events such as a Click event are handled by delegates that wrap methods (event handlers) on the page itself. So you usually end up with a single page that encapsulates all of the controls for the page, is responsible for displaying it, and handles all events associated with the page.

In JSF you associate events such as Click events with action or actionListener methods in a backing bean via method binding expressions. These expressions is written in JSF-EL (Expression Language) which also is used in value binding expressions where the value of e.g. a input filed is bound to a property in the bean.

Even though they differ in the semantics they both provide a very convenient way to hook up UI components to methods that perform the requested action. This abstraction has made web development much easier.

Navigation

Both ASP.NET and JSF have the possibility to externalize the navigation to an XML file. In ASP.NET you define a sitemap file as described in section 3.2.5 which can be used from different controls to handle navigation. In JSF you define navigation rules and outcomes in faces-config.xml which maps logical outcomes such as "view_products" to a physical view (e.g. a JSP page). These logical outcomes can then be returned from action methods in a backing bean to show the correct view. There is no hard-coded url redirecting from the backing bean code. In ASP.NET however, you often see hard-coded navigation in the methods in the code-behind file with the use of `Server.Transfer("somefile.aspx").` This can make a strong dependency between different pages, which is not always desirable if change in the application is to be implemented.

Templating

The ability to make consistent looking web applications is very powerful with the new master page feature in ASP.NET as described in section 3.2.4. It makes it easy to reuse and include content on different pages. JSF does not have a template mechanism beside the <jsp:include> element. This is okay for very simple reuse of content, but JSF misses another mechanism. However, other open-source J2EE frameworks provide this functionality like the Jakarta Struts Tiles project or OpenSymphony's SiteMesh framework.

Validation

A part of any application that process input from the user validation is often required to ensure that input have been entered in the right format, the user has selected something in a dropdown menu etc. Both ASP.NET and JSF provides validation controls / components that can be associated with the different input controls. The key difference is that ASP.NET support the use of client-side validation where JSF only supports server-side. This requires that JSF has to do a roundtrip to the server every time the user clicks a button and values should be validated. This can result in performance and user experience disadvantages in our opinion.

Rendering

ASP.NET components render themselves directly on the page. This is different from JSF where components can render themselves, but more often they delegate rendering to special Renderer objects made available in a RenderKit. A different RenderKit can be supplied for each different type of medium like html, wap etc. A default HTML RenderKit is provided by every JSF implementation. But this means that the same JSF component can be rendered differently in a web browser or wireless device by simply changing to a different RenderKit. This is a powerful capability and leads to the general principle in JSF about extensibility of the framework

Extensibility / plugability

Since JSF is a specification and not a product developed by one vendor, it has been designed to make it possible for vendors and the application developer himself to provide alternative and powerful implementations and plug in new features to the framework. This means that most of the framework can be extended and switched to support different things such as providing another ViewHandler or NavigationHandler. As a result you can e.g. use another view technology than JSP like the open-source alternative Facelets. Although the extensibility in JSF is a great feature it may not be as relevant regarding ASP.NET because there is only one vendor.

IDE support

As a final point we wish to elaborate on the IDE / tool support in the two technologies. Both ASP.NET and JSF was designed with tool support in mind, but the IDEs differ much in features and ease of use. ASP.NET has a very nice IDE in Visual Studio which has great functionalities regarding drag 'n' drop and easy and visual preview of pages, wizards for configuring the controls etc. We think that it is a little harder to find such an integrated IDE for JSF, although IDEs like Oracle JDeveloper and Sun's Java Studio Creator is very competitive. But the choice of IDE is also tight coupled to the choice of J2EE application server since they often is highly integrated. You may loose some features if you develop in JDeveloper and deploys your web application to the JBoss Application Server instead of Oracle Application Server.

6.6.2 Business logic layer

When it comes to the business logic layer J2EE and .NET both provides enterprise services such as security, transaction support, connection pooling, message queuing etc. Although these services are alike and almost delivers the same functionality their implementation is quite different. As described in section 3.3.1 the .NET framework offers these services primarily through Component Services (COM+).

J2EE on the other hand offer the Enterprise Java Bean (EJB) server-side component architecture. EJBs are the roughly equivalent of COM+ components or managed code components, and can provide services/cross-cutting concerns such as maintaining state, transactional support, method level security, message queue support and persistence support.

EJBs are deployed to a J2EE Application Server such as IBM WebSphere and are encapsulated in a component container. The container is responsible for receiving client requests and mediates the calls to the EJB in order to control the cross-cutting concerns. The EJBs properties such as transaction level, JNDI name and security options are configured through an XML deployment descriptor or as Java annotations on classes and methods (the latter only in EJB 3.0).

EJBs exist in three different forms:

- Session beans (stateless and stateful)
- Message-driven beans
- Entity beans

Session beans is used to model business logic and often provides some kind of business critic transactions or services such as retrieving all customers from a persistent storage. Session beans can be stateless which means they can't maintain state across method calls and their instances exists in object pools. They are often used to model a Session Facade [26] as a gateway to the server-side application and business logic. Stateful session beans maintain state across multiple method calls, tying the client to the session bean for the object's lifetime. This makes it possible to e.g. model a shopping cart in a web application which has to maintain state across requests.

Message-driven beans (MDB) are used in connection with JMS (Java Message Service) to process messages asynchronously. MDBs are JMS message consumers and clients do not access MDBs directly; they send a JMS message to a destination (JMS Server) where the MDB is listening. This enables reuse of the MDB, and also means that the developer does not have to worry about where to go to get the message. MDBs are like session beans used to implement business logic.

Entity beans are used to give an object-oriented view on relational data. In general you can say that they represent a domain object in persistent storage. In EJB 3.0 entity beans provide almost the same functionality as the object-relational mapping (ORM) tool Hibernate and are POJOs (Plain Old Java Objects). This means that they are normal Java objects that can be detached from the container, serialized and transferred all the way to the client. In that sense they replace the Data Transfer Object [8]. In the following section about the data access layer in J2EE and .NET we will shortly describe entity bean because they play a vital role in J2EE persistence.

6.6.3 Data access layer

Both J2EE and .NET contain data access libraries to access databases and other data sources. The core features of each library allow similar direct database access by opening a connection, submitting a command and receiving the results.

In .NET this task is accomplished using ADO.NET with SqlAdapters, DataReaders, DataSets etc. as described in section 3.4.1. ADO.NET is based on a disconnected model because of the DataSet and supports optimistic concurrency and the ability to read and write XML data. ADO.NET furthermore has the Strongly Typed DataSets as described in section 3.4.1.

In J2EE (or just J2SE actually) fundamental database access is accomplished with JDBC (Java Database Connectivity). JDBC is a set of interfaces that vendors have to implement in order to provide a JDBC driver for their database. This means that JDBC is not a product but a specification as the rest of J2EE. The JDBC library supports a very traditional approach to database access. The user submits a Statement (usually a query or stored procedure call) and may receive results in the form of a ResultSet with a live connection to the database. However, JDBC also supports disconnected result sets called CachedRowSets that supports serialization and optimistic concurrency.

JDBC does not directly support reading or writing XML data as ADO.NET. Although several vendors have created JDBC wrappers for XML-based database access, the core J2EE classes do not support it [12].

As mentioned in section 6.6.2 J2EE also provides **entity beans.** Entity beans represent an object in persistent storage. Each instance of an entity bean corresponds to a row of data from a database table, although it is not always that simple because of different mapping approaches between the relational and object-oriented world. There are two types of entity beans; container managed persistence (**CMP**) beans, which are managed by the container; and bean managed persistence (**BMP**) beans, which manage their own persistence (the programmer).

With CMP entity beans, the container manages the mapping between bean and database fields using a XML deployment descriptor or by reading Java annotations on the bean class. The container manages all the communications with the database and no database code is needed from the programmer of the bean.

With BMP entity beans, the developer is responsible for writing all the JDBC code for transferring data between a bean and the data store. BMPs require the developer to write much more data related code than CMP but in this way optimization and flexibility can be achieved.

As part of the entity beans specification is an object-oriented query language called EJB-QL. This is used to perform queries on the entity beans that at runtime are translated into the corresponding SQL calls targeted for the database in use. This helps enforce maintainability and makes it possible to switch the underlying database very easily. You can however also use native SQL queries on the entity beans to optimize performance or to use specialized features in your database system.

6.7 Summary of J2EE versus .NET

Throughout this section we have compared some of the features and technologies in J2EE and .NET. To give an overview we have listed the features and technologies in Table 1 on the next page (freely transformed and updated from [12]).

Feature or Service	Microsoft .NET	J2EE	Comments
Technology Type	Product	Standard/Specification	
Middleware Vendors	Microsoft and partners	50+ vendors	
Client Side GUI	Windows Forms	AWT/SWING	Part of J2SE
Web GUI	ASP.NET	Servlets /JSP and JavaServer Faces	
Web Scripting	ISAPI HttpHandler HttpModule	Servlet Filter	
Web Application Hosting	Internet Information Server	Multiple (depends on vendor)	J2EE examples include Apache Tomcat
Runtime engine	CLR (Common Language Runtime)	JRE (Java Runtime Engine)	
Server Side Business Logic Component	.NET Class or Serviced Component (COM+)	EJB Session Beans	
Server Side Data Components 1	Serviced Component with DB Logic	EJB with Bean Managed Persistence	
Server Side Data Components 2	ADO.NET Data Set	EJB with Container Managed Persistence	Only an approximate equivalence!
Directory Access	Active Directory Services Interface (ADSI) through LDAP	Java Naming and Directory Service (JNDI) through LDAP	LDAP compatibility makes switching between directory services very easy.
Remote Invocation	.NET Remoting	RMI-IIOP	
Data Access	ADO.NET	JDBC, SQL/J, JDO	
Messaging	Microsoft Message Queuing	JMS (Java Message Service)	Microsoft Message Queuing is a product. JMS is a specification
Transactional Support	COM+/Distributed Transaction Controller (DTC)	JTA (Java Transaction API)	

Table 1 - J2EE versus .NET

7 Evaluation of ASP.NET 2.0

ASP.NET 2.0 and .NET in general is a powerful and well-supported software development framework and it has been both interesting and educational to work with various aspects of this technology. One of the strengths of the .NET framework is that it is based on XML, which facilitates communication between different devices and software systems. It has thus been interesting to see how web services can be used to create communication paths between different systems. Another great advantage of the .NET framework is the tool support integrated in Visual Studio. With Visual Studio it is e.g. easy to create functional GUIs for both standard applications and for websites. Normally this simplicity is partially a result of the direct data binding that is established. In our case we have implemented a middle tier that all the control is bound to and thereby removed the direct data binding from the presentation controls to the database. This was actually quite easy due to the new ObjectDataSource control in ASP.NET 2.0.

The ADO.NET technology provides a powerful and efficient tool for linking the applications to a database and managing concurrency issues, even that this was not used in the implementation. In relation to the data access layer, we have also experimented with NHibernate that seemed quite nice to work with.

Much of the efficiency when working in Visual Studio comes from the fact that most of the application code is auto generated. Besides the efficiency this has the advantage that it is ensured that these code parts are constructed in a uniform fashion. It does, however, also have a disadvantage, namely that the developer only has direct insight into a smaller part of the code.

Overall, we find that the .NET framework is very well suited for developing business and enterprise applications efficiently, and we are convinced that the .NET technologies will continue to be a great alternative or primary choice in regards to J2EE. As a final remark we think that it is noteworthy to mention the good effect of the increased competition in the field of enterprise development frameworks. .NET really pushes the technology further and forces the other competitors like J2EE to develop even more competitive enterprise application frameworks.

8 Conclusion

As stated in the problem definition the purpose of this project has been to discuss architecture and application design of object-oriented distributed applications in general and in relation to .NET. Differences between J2EE and .NET in relation to these subjects were going to be examined. Further based on our studies a small sample application that illustrates blueprints of a .NET-based layered application architecture was going to be implemented. Our intention was also to learn about features in ASP.NET 2.0 and Visual Studio 2005.

The scope of this project has been wide. In order to answer the problem definition we have conducted a lot of literature studies and in order to write this report we have also conducted the following development activities: software architecture, database design, database implementation, network configuration and we have worked with a wide range of technologies/tools: XML, C#, Visual Studio, CVS, SQL Server, ASP.NET and ADO.NET. The size of this project has at times been both frustrating and overwhelming, but we have certainly learned a lot by completing the project. It has been very satisfying to explore layered application design in the .NET framework and especially ASP.NET 2.0. Further the comparison with J2EE has provided us with a great overview of the similarities and differences between the two frameworks. The comparison also provided us with great insight about what enterprise application frameworks are able to provide today.

Based on the fact that we have fulfilled the goals of the project and also learned about new features in ASP.NET 2.0, the C# programming language through our implementation of the sample application, the .NET environment and Visual Studio 2005 we conclude that the project has been successful.

9 Perspectives

Even though the wide scope of this project have allowed us to examine lots of different features in .NET and to some extend J2EE there are still many interesting areas left out. Regarding the data access layer some pretty exiting things is happening around C# 3.0 that includes its new Language Integrated Query (LINQ), which provides a standard way of querying non-object-oriented from within the .NET languages. In this project we have discussed layered architecture. In relation to the data access layer it could be interesting to examine what effect LINQ will have on the application design.

In relation to the presentation layer new exciting things are also happening. In the web context a new paradigm is evolving with the AJAX technology. It could be interesting to investigate AJAX in relation to ASP.NET and JavaServer Faces. Both have some initiatives going on in the area and especially Microsoft is promoting their version called ATLAS. The idea of using Asynchronous JavaScript and XML in web applications will certainly enrich the user experience on the web. Another UI related technology from Microsoft for developing rich clients is Windows Presentation Foundation that is launched with Windows Vista. This is going to replace the current Windows Forms applications and provides some really interesting features because of 3D acceleration, vector graphics and the possibility to declarative build your interfaces in XAML.

As a final remark it could be really interesting to examine how SOA is going to change application architecture and communication between processes. Also here Microsoft is launching a new platform together with Windows Vista called Windows Communication Foundation. This is a new API that unifies the former technologies in .NET like Remoting and XML Web Services to provide a foundation for the future SOA-based applications.

10 Bibliography

Magazine articles:

[1] Esposito, Dino
DataSets vs. Collections
MSDN Magazine, August 2005

[2] John Papa
Efficient Coding With Strongly Typed DataSets
MSDN Magazine, December 2004

[3] Prosise, Jeff
An Overview Of The New Services, Controls, And Features In ASP.NET 2.0
MSDN Magazine, 2006

[4] Prosise, Jeff.
Asynchronous Pages in ASP.NET 2.0
MSDN Magazine, October 2005

Books:

[5] Crocker, Angela., Olsen, Andy. and Jezierski, Edward
Designing Data Tier Components and Passing Data Through Tiers
Microsoft patterns & practices aticle, August 2002
URL http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/BOAGag.asp
Last accessed 2006-05-17

[6] Darie, Cristian. and Watson, Karli
Beginning ASP.NET 2.0 E-Commerce in C# 2005
Apress, 2006

[7] Esposito, Dino
Introducing Microsoft ASP.NET 2.0
Microsoft Press, 2005

[8] Fowler, Martin
Patterns of Enterprise Architecture
Addison-Wesley Professional, 2002

[9] Gamma, Erich., Helm, Richard., Johnson, Ralph., John Vlissides
Design Patterns: Elements of Reusable Object-Oriented Software
Addison-Wesley Professional Comuting series, January 15, 1995

[10] Glenn Johnson
Programming Mircrosoft ADO.NET 2.0 Applicationc Advanced Topics, 2005 edition
Microsoft Press, 2006

[11] Larman, Craig
Applying UML and Patterns: An introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd ed.
Pearson Education Inc., 2005

[12] Laudati, Peter., et al.
Application Interoperability: Microsoft .NET and J2EE
Microsoft pattern & practices, December 2003
URL http://msdn.microsoft.com/library/en-us/dnpag/html/JDNI.asp

[13] Selly, Dominic., Troelsen, Andres. and Tom Barnaby
Expert ASP.NET 2.0 Advanced Application Design
Apress, 2006

[14] Singh, Inderjeet., Stearns, Beth., Johnson, Mark. and the Enterprise Team
Designing Enterprise Applications with the J2EE(TM) Platform, Second Edition
Addison-Wesley, 2002

[15] Webb, Jeff
Developint XML Web Services and Server Componentes with Microsoft Visual Basic .NET and Visual C# .NET
Microsoft Press, 2003

Online articles:

[16] Chaffee, Alex
One, two, three, or n tiers
URL http://www.javaworld.com/javaworld/jw-01-2000/jw-01-ssj-tiers.html
Last accessed 2006-05-18

[17] Dobson, Rick
Using DataReaders to Increase Speed and Reduce Memory
URL http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnhcvs04/html/vs04g6.asp
Last accessed 2006-05-17

[18] Mortensen, M. Keith., McGovern, Rob., Liptaak, Charles
ASP.NET and Struts: Web Application Architectures
MSDN Library article, December 2003
URL http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspp/html/ASPNet-ASPNet-J2EE-Struts.asp
Last accessed 2006-05-24

[19] Sheriff, Paul D.
Building an N-Tier Application in .NET
MSDN Library article, February 2002
URL http://msdn.microsoft.com/library/en-us/dndotnet/html/buildntierapp.asp
Last accessed 2006-05-17

[20] Sheriff, Paul D.
Designing a .NET Application
MSDN Library article, April 2002
URL http://msdn.microsoft.com/library/en-us/dndotnet/html/designnetapp.asp
Last accessed 2006-05-17

[21] Rørbæk, Jeppe
ASP.NET 2.0 DataSourceControls og Databinding
URL download.microsoft.com/download/3/3/c/ 33c00d2e-05f9-4655-a35b-5e5356705a0e/TechTalk_DataSourceControls.ppt
Last accessed 2006-05-23

[22] Walther, Stephen
Working with the ASP.NET 2.0 ObjectDataSource Control
URL http://msdn.microsoft.com/asp.net/reference/data/default.aspx?pull=/library/en-us/dnvs05/html/asp2objectdatasource.asp
Last accessed 2006-05-23

World Wide Web:

[23] ASP.NET Master Pages Overview
URL http://msdn2.microsoft.com/en-US/library/wtxbf3hh.aspx
Last accessed 2006-05-23

[24] ASP.NET Data-Bound Web Server Controls Overview
URL http://msdn2.microsoft.com/en-US/library/ms228214.aspx
Last accessed 2006-05-23

[25] Brian Ritchie
Brian Ritchie's .NET Development Site
URL http://dotnetpowered.com/languages.aspx
Last accessed 2006-05-24

[26] Core J2EE Pattern Catalog
Session Facade
URL http://java.sun.com/blueprints/corej2eepatterns/Patterns/SessionFacade.html
Last accessed 2006-05-26

[27] Data Types Supported by XML Web Services Created Using ASP.NET
.NET Framework Developer's Guide
URL http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpcondatatypessupportedbywebservices.asp
Last accessed 2006-05-24

[28] Dinesh Chandnani, .Net Client Team
BindingSource and BindingList Of T - DataBinding made simple
URL http://blogs.msdn.com/dchandnani/archive/2005/03/12/394438.aspx
Last accessed 2006-05-24

[29] Komponentbaseret Design og J2EE lecture slides
URL http://www.itu.dk/courses/SKBD/F2006/lecture2.pdf
Last accessed 2006-05-26

[30] Le, Alan et al.
Microsoft .NET Pet Shop 4: Migrating an ASP.NET 1.1 Application to 2.0
Microsoft Corporation and Vertigo Software, Inc., February 2006
URL http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/bdasamppet4.asp
Last accessed 2006-05-22

[31] Managed/designs Northwind Starter Kit
URL http://download.manageddesigns.it/nsk.aspx
Last accessed 2006-05-22

[32] Microsoft Corporation
Data Access Application Block
Microsoft patterns & practices, January 2006
URL http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/entlibjan2006_dataaccessappblock.asp
Last accessed 2006-05-22

[33] Peter Sestoft
C# Project Cluster webpage
URL http://www.itu.dk/people/sestoft/csharp/
Last accessed 2006-04-29

[34] Wikipedia The Free Encyclopedia
http://en.wikipedia.org/wiki/ADO.NET
Last accessed 2006-05-22

11 Appendices

Appendix 1:

Overview Of The New Services, Controls, And Features In ASP.NET 2.0 not examined in the report.

Appendix 2:

Northwind as a strongly typed DataSet

Appendix 3:

Screenshots of WebForm and WinForm

Appendix 4:

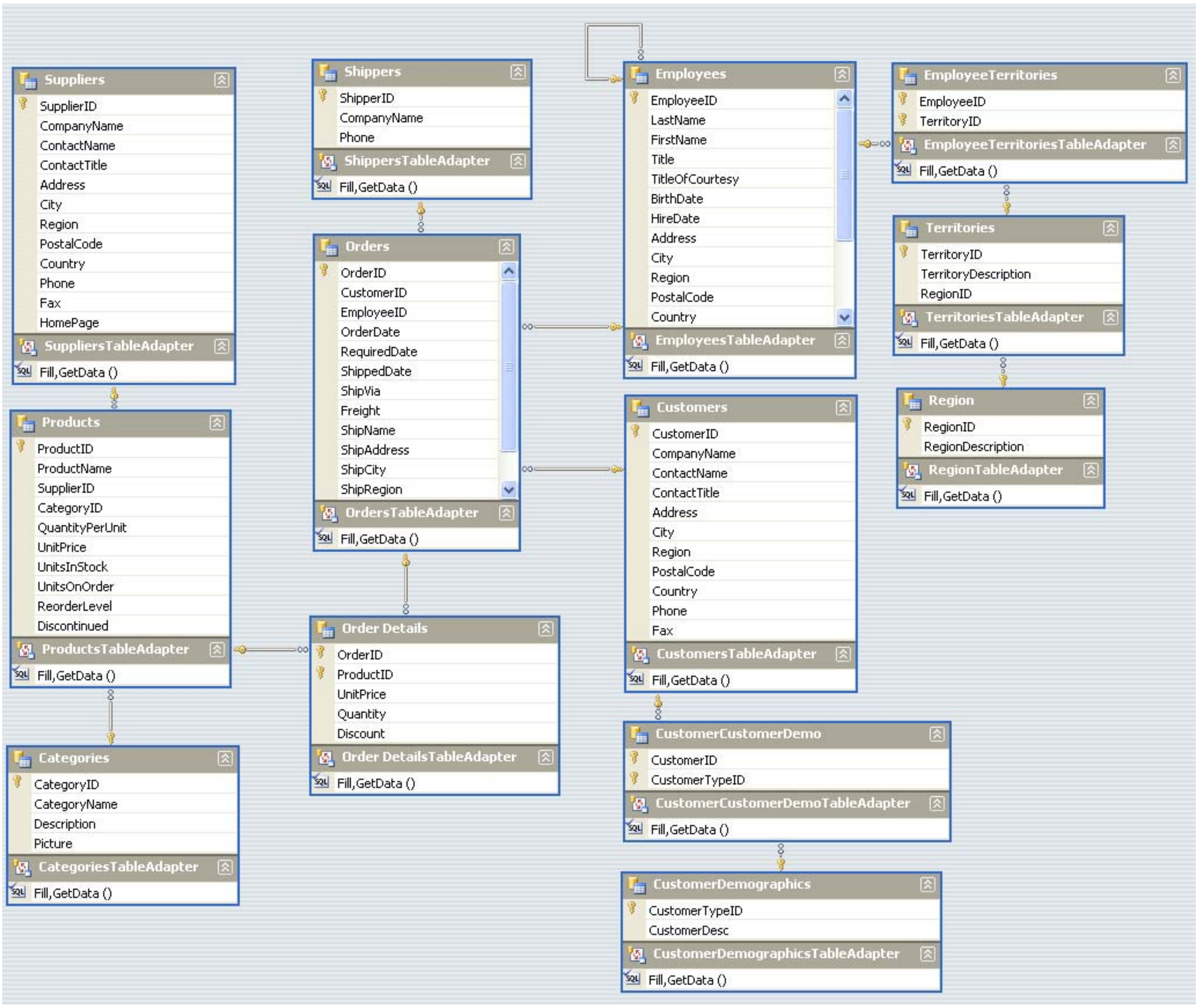
Source code

Appendix 1: Overview Of The New Services, Controls, And Features In ASP.NET 2.0 not examined in the report.

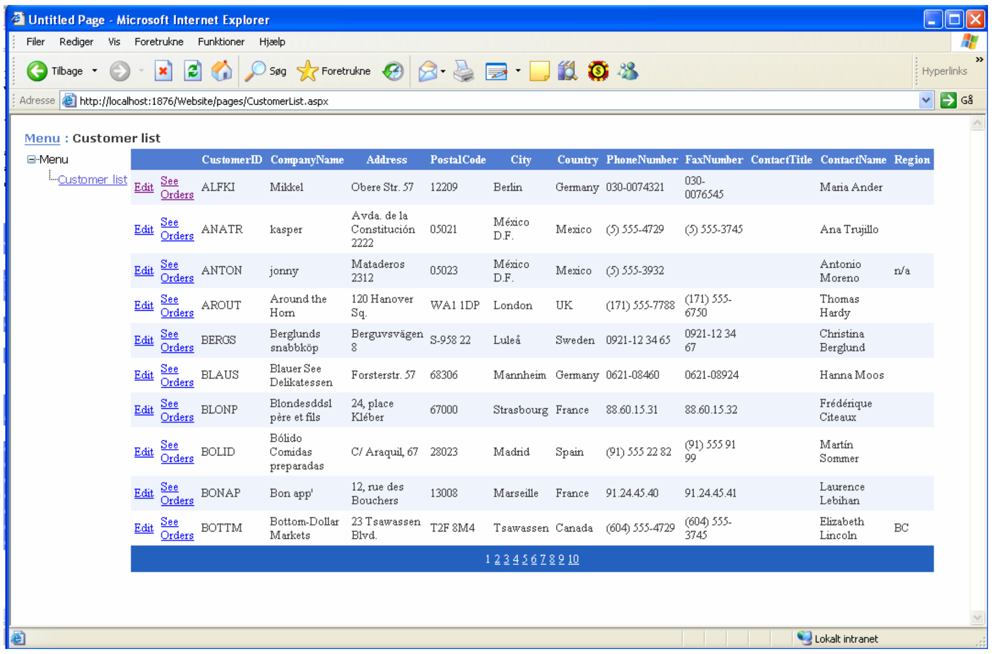
- New Dynamic Compilation Model
- Precompiling and Deploying Without Source
- Other New Controls
- Membership Service
- Login Controls
- Role Management
- Profiles
- URL Mapping
- SQL Cache Dependencies
- Validation Groups
- Cross-Page Postbacks
- Client Callback Manager (XML-HTTP)
- Asynchronous Pages
- Encrypted Configuration Sections

This is based on [3]

Appendix 2: Northwind as a strongly typed DataSet

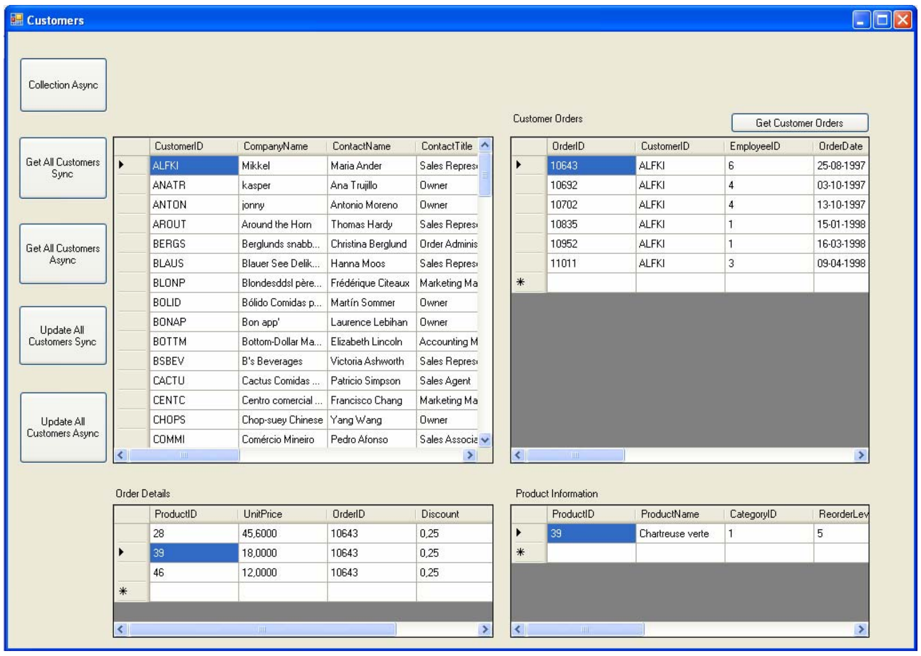


Appendix 3: Screenshots of WebForm and WinForm


Untitled Page - Microsoft Internet Explorer
Filer Rediger Vis Foretrukne Funktioner Hjælp
Tilbage
Søg
Foretrukne
Hyperlinks
Adresse
http://localhost:1876/Website/pages/CustomerList.aspx
Gå
Menu : Customer list
Menu
Customer list
Lokalt intranet

		CustomerID	CompanyName	Address	PostalCode	City	Country	PhoneNumber	FaxNumber	ContactTitle	ContactName	Region
Edit	See Orders	ALFKI	Mikkel	Obere Str. 57	12209	Berlin	Germany	030-0074321	030-0076545		Maria Ander	
Edit	See Orders	ANATR	kasper	Avda. de la Constitución 2222	05021	México D.F.	Mexico	(5) 555-4729	(5) 555-3745		Ana Trujillo	
Edit	See Orders	ANTON	jonny	Mataderos 2312	05023	México D.F.	Mexico	(5) 555-3932			Antonio Moreno	n/a
Edit	See Orders	AROUT	Around the Horn	120 Hanover Sq.	WA1 1DP	London	UK	(171) 555-7788	(171) 555-6750		Thomas Hardy	
Edit	See Orders	BERGS	Berglunds snabbköp	Berguvsvägen 8	S-958 22	Luleå	Sweden	0921-12 34 65	0921-12 34 67		Christina Berglund	
Edit	See Orders	BLAUS	Blauer See Delikatessen	Forsterstr. 57	68306	Mannheim	Germany	0621-08460	0621-08924		Hanna Moos	
Edit	See Orders	BLONP	Blondesddsl père et fils	24, place Kléber	67000	Strasbourg	France	88.60.15.31	88.60.15.32		Frédérique Citeaux	
Edit	See Orders	BOLID	Bólido Comidas preparadas	C/ Araquil, 67	28023	Madrid	Spain	(91) 555 22 82	(91) 555 91 99		Martín Sommer	
Edit	See Orders	BONAP	Bon app'	12, rue des Bouchers	13008	Marseille	France	91.24.45.40	91.24.45.41		Laurence Lebihan	
Edit	See Orders	BOTTM	Bottom-Dollar Markets	23 Tsawassen Blvd.	T2F 8M4	Tsawassen	Canada	(604) 555-4729	(604) 555-3745		Elizabeth Lincoln	BC

1 2 3 4 5 6 7 8 9 10



	CustomerID	CompanyName	ContactName	ContactTitle
▶	ALFKI	Mikkel	Maria Ander	Sales Repres
	ANATR	kasper	Ana Trujillo	Owner
	ANTON	jonny	Antonio Moreno	Owner
	AROUT	Around the Horn	Thomas Hardy	Sales Repres
	BERGS	Berglunds snabb...	Christina Berglund	Order Adminis
	BLAUS	Blauer See Delik...	Hanna Moos	Sales Repres
	BLONP	Blondesddsl père...	Frédérique Citeaux	Marketing Ma
	BOLID	Bólido Comidas p...	Martín Sommer	Owner
	BONAP	Bon app'	Laurence Lebihan	Owner
	BOTTM	Bottom-Dollar Ma...	Elizabeth Lincoln	Accounting M
	BSBEV	B's Beverages	Victoria Ashworth	Sales Repres
	CACTU	Cactus Comidas ...	Patricio Simpson	Sales Agent
	CENTC	Centro comercial ...	Francisco Chang	Marketing Ma
	CHOPS	Chop-suey Chinese	Yang Wang	Owner
	COMMI	Comércio Mineiro	Pedro Afonso	Sales Associa

Customer Orders

	OrderID	CustomerID	EmployeeID	OrderDate
▶	10643	ALFKI	6	25-08-1997
	10692	ALFKI	4	03-10-1997
	10702	ALFKI	4	13-10-1997
	10835	ALFKI	1	15-01-1998
	10952	ALFKI	1	16-03-1998
	11011	ALFKI	3	09-04-1998
*				

Order Details

	ProductID	UnitPrice	OrderID	Discount
	28	45,6000	10643	0,25
▶	39	18,0000	10643	0,25
	46	12,0000	10643	0,25
*				

Product Information

	ProductID	ProductName	CategoryID	ReorderLev
▶	39	Chartreuse verte	1	5
*				

Appendix 4: Source code

The full source code is included from the next page and forward.

ITU.BusinessEntities

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ITU.BusinessEntities
{
    [Serializable]
    public class Customer
    {
        private string customerID = string.Empty;
        private string companyName = string.Empty;
        private string contactName = string.Empty;
        private string contactTitle = string.Empty;
        private string address = string.Empty;
        private string city = string.Empty;
        private string region = string.Empty;
        private string postalCode = string.Empty;
        private string country = string.Empty;
        private string phone = string.Empty;
        private string fax = string.Empty;


        //private string originalcustomerID = string.Empty;
        //private string originalCompanyName = string.Empty;
        //private string originalContactName = string.Empty;
        //private string originalContactTitle = string.Empty;
        //private string originalAddress = string.Empty;
        //private string originalCity = string.Empty;
        //private string originalRegion = string.Empty;
        //private string originalPostalCode = string.Empty;
        //private string originalCountry = string.Empty;
        //private string originalPhoneNumber = string.Empty;
        //private string originalFaxNumber = string.Empty;

        public Customer() { }

        public Customer(string customerID, string companyName, string contactName, 
    string contactTitle,string address, string city, string region, string postalCode 
    , string country, string phone, string fax)
        {
            this.customerID = customerID;
            this.companyName = companyName;
            this.contactName = contactName;
            this.contactTitle = ContactTitle;
            this.address = address;
            this.city = city;
            this.region = region;
            this.postalCode = postalCode;
            this.country = country;
            this.phone = phone;
            this.fax = fax;
        }

        public string CustomerID
        {
            get { return customerID; }
            set { customerID = value; }
        }

        public string CompanyName
        {
            get { return companyName; }
            set { companyName = value; }
        }

        public string ContactName
        {
            get { return contactName; }
            set { contactName = value; }
        }

        public string ContactTitle
        {
            get { return contactTitle; }
            set { contactTitle = value; }
        }

        public string Address
        {
            get { return address; }
            set { address = value; }
        }

        public string City
        {
            get { return city; }
            set { city = value; }
        }

        public string Region
        {
            get { return region; }
            set { region = value; }
        }

        public string PostalCode
        {
            get { return postalCode; }
            set { postalCode = value; }
        }

        public string Country
        {
            get { return country; }
            set { country = value; }
        }

        public string Fax
        {
            get { return fax; }
            set { fax = value; }
        }

        public string Phone
        {
            get { return phone; }
            set { phone = value; }
        }
    }
}

```

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ITU.BusinessEntities
{
    public class Product
    {
        private string productID = string.Empty;
        private string productName = string.Empty;
        private string supplierID = string.Empty;
        private string categoryID = string.Empty;
        private string quantityPerUnit = string.Empty;
        private string unitPrice = string.Empty;
        private string unitsInStock = string.Empty;
        private string unitsOnOrder = string.Empty;
        private string reorderLevel = string.Empty;
        private string discontinued = string.Empty;

        public Product()
        {
        }

        public Product(string productID, string productName, string supplierID, string 
     categoryID, string quantityPerUnit, string unitPrice, string unitsInStock, string 
     unitsOnOrder, string reorderLevel, string discontinued)
        {
            this.productID = productID;
            this.productName = productName;
            this.supplierID = supplierID;
            this.categoryID = categoryID;
            this.quantityPerUnit = quantityPerUnit;
            this.unitPrice = unitPrice;
            this.unitsInStock = unitsInStock;
            this.unitsOnOrder = unitsOnOrder;
            this.reorderLevel = reorderLevel;
            this.discontinued = discontinued;
        }

        public string ProductID
        {
            get { return productID; }
            set { productID = value; }
        }

        public string ProductName
        {
            get { return productName; }
            set { productName = value; }
        }

        public string SupplierID
        {
            get { return supplierID; }
            set { supplierID = value; }
        }

        public string CategoryID
        {
            get { return categoryID; }
            set { categoryID = value; }
        }

        public string QuantityPerUnit
        {
            get { return quantityPerUnit; }
            set { quantityPerUnit = value; }
        }

        public string UnitPrice
        {
            get { return unitPrice; }
            set { unitPrice = value; }
        }

        public string UnitsInStock
        {
            get { return unitsInStock; }
            set { unitsInStock = value; }
        }

        public string UnitsOnOrder
        {
            get { return unitsOnOrder; }
            set { unitsOnOrder = value; }
        }

        public string ReorderLevel
        {
            get { return reorderLevel; }
            set { reorderLevel = value; }
        }

        public string Discontinued
        {
            get { return discontinued; }
            set { discontinued = value; }
        }
    }
}
```

```csharp
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;

namespace ITU.BusinessEntities
{
    public class Order
    {
        private string orderID = string.Empty;
        private string customerID = string.Empty;
        private string employeeID = string.Empty;
        private string orderDate = string.Empty;
        private string requiredDate = string.Empty;
        private string shippedDate = string.Empty;
        private string shipVia = string.Empty;
        private string freight = string.Empty;
        private string shipName = string.Empty;
        private string shipAddress = string.Empty;
        private string shipCity = string.Empty;
        private string shipRegion = string.Empty;

        public Order()
        {
        }

        public Order(string orderID, string customerID, string employeeID, string orderDate, string requiredDate, string shippedDate, string shipVia, string freight, string shipName, string shipAddress, string shipCity, string shipRegion)
        {
            this.orderID = orderID;
            this.customerID = customerID;
            this.employeeID = employeeID;
            this.freight = freight;
            this.orderDate = orderDate;
            this.orderID = orderID;
            this.shipAddress = shipAddress;
            this.shipName = shipName;
            this.shipCity = shipCity;
            this.shippedDate = shippedDate;
            this.shipVia = shipVia;
            this.shipRegion = shipRegion;
        }

        public string OrderID
        {
            get { return orderID; }
            set { orderID = value; }
        }

        public string CustomerID
        {
            get { return customerID; }
            set { customerID = value; }
        }

        public string EmployeeID
        {
            get { return employeeID; }
            set { employeeID = value; }
        }

        public string OrderDate
        {
            get { return orderDate; }
            set { orderDate = value; }
        }

        public string RequiredDate
        {
            get { return requiredDate; }
            set { requiredDate = value; }
        }

        public string ShippedDate
        {
            get { return shippedDate; }
            set { shippedDate = value; }
        }

        public string ShipVia
        {
            get { return shipVia; }
            set { shipVia = value; }
        }

        public string Freight
        {
            get { return freight; }
            set { freight = value; }
        }

        public string ShipName
        {
            get { return shipName; }
            set { shipName = value; }
        }

        public string ShipAddress
        {
            get { return shipAddress; }
            set { shipAddress = value; }
        }

        public string ShipCity
        {
            get { return shipCity; }
            set { shipCity = value; }
        }

        public string ShipRegion
        {
            get { return shipRegion; }
            set { shipRegion = value; }
        }
    }
}

```

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ITU.BusinessEntities
{
    public class OrderDetail
    {
        private string orderID = string.Empty;
        private string productID = string.Empty;
        private string unitPrice = string.Empty;
        private string quantity = string.Empty;
        private string discount = string.Empty;

        public OrderDetail()
        {
        }

        public OrderDetail(string orderID, string productID, string unitPrice, string ↙
    quantity, string discount)
        {
            this.orderID = orderID;
            this.productID = productID;
            this.unitPrice = unitPrice;
            this.quantity = quantity;
            this.discount = discount;
        }

        public string OrderID
        {
            get { return orderID; }
            set { orderID = value; }
        }

        public string ProductID
        {
            get { return productID; }
            set { productID = value; }
        }

        public string UnitPrice
        {
            get { return unitPrice; }
            set { unitPrice = value; }
        }

        public string Quantity
        {
            get { return quantity; }
            set { quantity = value; }
        }

        public string Discount
        {
            get { return discount; }
            set { discount = value; }
        }
    }
}
```

ITU.DAL

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ITU.DAL
{
    public interface IDataProviderFactory
    {
        ICustomerDataProvider GetCustomerProvider();
        IOrderDataProvider GetOrderProvider();
        IProductDataProvider GetProductProvider();
    }
}
```

```csharp
using System;
using System.Collections.Generic;
using System.Text;
using ITU.BusinessEntities;
using System.Data;

namespace ITU.DAL
{
    public interface IOrderDataProvider
    {
        void Create(Order o);
        void Update(Order o);
        void Delete(Order o);

        Order GetById(string orderId);

        DataSet GetOrdersDataSet();
        void UpdateOrdersDataSet(DataSet ds);

        IList<Order> GetOrders(string customerId);
        IList<OrderDetail> GetOrderDetails(string orderId);
    }
}

```

```
using System;
using System.Collections.Generic;
using System.Text;
using ITU.BusinessEntities;
using System.Data;

namespace ITU.DAL
{
    public interface ICustomerDataProvider
    {
        void Create(Customer c);
        void Update(Customer c);
        void Delete(Customer c);
        Customer GetById(string customerId);

        DataSet GetCustomersDataSet();
        void UpdateCustomersDataSet(DataSet ds);

        IList<Customer> GetCustomers();
    }
}

```

```
using System;
using System.Collections.Generic;
using System.Text;
using ITU.BusinessEntities;

namespace ITU.DAL
{
    public interface IProductDataProvider
    {
        Product GetById(string productId);
        IList<Product> GetProducts(string[] productId);
    }
}

```

ITU.DAL.Factory

```csharp
using System;
using System.Collections.Generic;
using System.Text;
using System.Configuration;
using System.Reflection;
using ITU.DAL;

namespace ITU.DAL.Factory
{
    public class DataAccessProviderFactory : IDataProviderFactory
    {
        private static Assembly activeDataProvider = null;
        private static IDataProviderFactory activeDataProviderFactory = null;
        private static ICustomerDataProvider customerDataProvider = null;
        private static IOrderDataProvider orderDataProvider = null;
        private static IProductDataProvider productDataProvider = null;

        static DataAccessProviderFactory()
        {
            string providerName = ConfigurationSettings.AppSettings["DataProvider"];
            string providerFactoryName = ConfigurationSettings.AppSettings["DataProviderFactory"];
            activeDataProvider = Assembly.Load(providerName);
            activeDataProviderFactory = (IDataProviderFactory)activeDataProvider.CreateInstance(providerFactoryName);
        }

        ICustomerDataProvider IDataProviderFactory.GetCustomerProvider()
        {
            if (customerDataProvider != null)
                return customerDataProvider;
            else
            {
                customerDataProvider = activeDataProviderFactory.GetCustomerProvider();
                return customerDataProvider;
            }
        }

        IOrderDataProvider IDataProviderFactory.GetOrderProvider()
        {
            if (orderDataProvider != null)
                return orderDataProvider;
            else
            {
                orderDataProvider = activeDataProviderFactory.GetOrderProvider();
                return orderDataProvider;
            }
        }

        IProductDataProvider IDataProviderFactory.GetProductProvider()
        {
            if (productDataProvider != null)
                return productDataProvider;
            else
            {
                productDataProvider = activeDataProviderFactory.GetProductProvider();
                return productDataProvider;
            }
        }
    }
}

```

ITU.DAL.SqlServer

```
using System;
using System.Collections.Generic;
using System.Text;
using ITU.DAL;

namespace ITU.DAL.SqlServer
{
    class DataProviderFactory : IDataProviderFactory
    {

        public ICustomerDataProvider GetCustomerProvider()
        {
            return new CustomerDataProvider();
        }

        public IOrderDataProvider GetOrderProvider()
        {
            return new OrderDataProvider();
        }

        public IProductDataProvider GetProductProvider()
        {
            return new ProductDataProvider();
        }
    }
}

```

```
using System;
using System.Collections;
using System.Collections.Specialized;
using System.Configuration;

namespace ITU.DAL.SqlServer
{
    internal class DataProviderHelper
    {
        private static string ConfigNode = "ITU.DAL.SqlServer";
        private static string connectionString = string.Empty;

        static DataProviderHelper()
        {
            NameValueCollection values = (NameValueCollection) ConfigurationSettings. ↙
    GetConfig(ConfigNode);
            connectionString = values["ConnectionString"];
        }

        public static string ConnectionString
        {
            get { return connectionString; }
        }
    }
}
```

```csharp
using System;
using System.Collections.Generic;
using System.Text;
using ITU.DAL;
using ITU.BusinessEntities;
using System.Data.SqlClient;
using System.Data;

namespace ITU.DAL.SqlServer
{
    public class OrderDataProvider : IOrderDataProvider
    {
        private static SqlConnection myConnection = new SqlConnection(DataProviderHelper.ConnectionString);

        public void Create(Order o)
        {
        }

        public void Update(Order o)
        {
        }

        public void Delete(Order o)
        {
        }

        public Order GetById(string orderId)
        {
            return new Order();
        }

        public DataSet GetOrdersDataSet()
        {
            return new DataSet();
        }

        public void UpdateOrdersDataSet(DataSet ds)
        {
        }

        public IList<OrderDetail> GetOrderDetails(string orderId)
        {
            IList<OrderDetail> orderDetails = new List<OrderDetail>();

            try
            {
                string query = String.Format("SELECT OrderID, ProductID, UnitPrice, Quantity, Discount FROM \"Order Details\" WHERE OrderID='{0}'", orderId);
                SqlCommand myCommand = new SqlCommand(query, myConnection);

                myConnection.Open();

                SqlDataReader myReader;
                myReader = myCommand.ExecuteReader();
                // Always call Read before accessing data.

                while (myReader.Read())
                {
                    orderDetails.Add(new OrderDetail(
                        myReader["OrderID"].ToString(),
                        myReader["ProductID"].ToString(),
                        myReader["UnitPrice"].ToString(),
                        myReader["Quantity"].ToString(),
                        myReader["Discount"].ToString()
                        ));
                }
                // always call Close when done reading.
                myReader.Close();
                // Close the connection when done with it.
                myConnection.Close();
            }
            finally
            {
                myConnection.Close();
            }

            return orderDetails;
        }




        public IList<Order> GetOrders(string customerId)
        {
            IList<Order> orders = new List<Order>();

            try
            {
                string query = String.Format("SELECT OrderID, CustomerID, EmployeeID, ↙
     OrderDate, RequiredDate, ShippedDate, ShipVia, Freight, ShipName, ShipAddress,   ↙
    ShipCity, ShipRegion FROM Orders WHERE CustomerID='{0}'", customerId);
                SqlCommand myCommand = new SqlCommand(query, myConnection);

                myConnection.Open();

                SqlDataReader myReader;
                myReader = myCommand.ExecuteReader();
                // Always call Read before accessing data.

                while (myReader.Read())
                {
                    orders.Add(new Order(
                        myReader["OrderID"].ToString(),
                        myReader["CustomerID"].ToString(),
                        myReader["EmployeeID"].ToString(),
                        myReader["OrderDate"].ToString(),
                        myReader["RequiredDate"].ToString(),
                        myReader["ShippedDate"].ToString(),
                        myReader["ShipVia"].ToString(),
                        myReader["Freight"].ToString(),
                        myReader["ShipName"].ToString(),
                        myReader["ShipAddress"].ToString(),
                        myReader["ShipCity"].ToString(),
                        myReader["ShipRegion"].ToString()
                        ));
                }
                // always call Close when done reading.
                myReader.Close();
                // Close the connection when done with it.
                myConnection.Close();
            }
            finally
            {
                myConnection.Close();
            }

            return orders;
        }
    }
}
```

```csharp
using System;
using System.Collections.Generic;
using System.Text;
using ITU.DAL;
using ITU.BusinessEntities;
using System.Data.SqlClient;
using System.Data;

namespace ITU.DAL.SqlServer
{
    public class CustomerDataProvider : ICustomerDataProvider
    {
        private static readonly string updateCustomerText = "UPDATE Customers SET CompanyName=@CompanyName, ContactName=@ContactName, ContactTitle=@ContactTitle, Address=@Address, City=@City, Region=@Region, PostalCode=@PostalCode, Country=@Country, Phone=@Phone, Fax=@Fax WHERE CustomerId=@CustomerId";
        private static SqlConnection conn = new SqlConnection(DataProviderHelper.ConnectionString);

        public void Create(Customer c)
        {
            throw new Exception("The method or operation is not implemented.");
        }

        public void Update(Customer c)
        {
            SqlCommand myCommand = new SqlCommand();

            myCommand.Connection = conn;
            myCommand.CommandText = updateCustomerText;
            myCommand.Parameters.Add(GetVariablesByEntity(c));

            conn.Open();
            myCommand.ExecuteNonQuery();
            conn.Close();
        }

        public void Delete(Customer c)
        {
            string query = String.Format("DELETE FROM Customers WHERE CustomerID = '{0}'", c.CustomerID);

            SqlCommand myCommand = new SqlCommand(query, conn);
            conn.Open();
            myCommand.ExecuteNonQuery();
            conn.Close();
        }

        public IList<Customer> GetCustomers()
        {
            IList<Customer> customers = new List<Customer>();

            try
            {
                string query = "SELECT customerID, companyName, contactName, contactTitle, address, city, region, postalCode, country, phone, fax FROM Customers";
                SqlCommand myCommand = new SqlCommand(query, conn);

                conn.Open();

                SqlDataReader myReader;
                myReader = myCommand.ExecuteReader();
                // Always call Read before accessing data.

                while (myReader.Read())
                {
                    customers.Add(new Customer(
                                myReader["customerID"].ToString(),
                                myReader["companyName"].ToString(),
                                myReader["contactName"].ToString(),
                                myReader["contactTitle"].ToString(),
                                myReader["address"].ToString(),
                                myReader["city"].ToString(),
                                myReader["region"].ToString(),
                                myReader["postalCode"].ToString(),
                                myReader["country"].ToString(),
                                myReader["phone"].ToString(),
                                myReader["fax"].ToString()
                                ));
                }
                // always call Close when done reading.
                myReader.Close();
                // Close the connection when done with it.
                conn.Close();
            }
            finally
            {
                conn.Close();
            }

            return customers;
        }

        public DataSet GetCustomersDataSet()
        {
            string query = "SELECT * FROM Customers";
            SqlDataAdapter sda = new SqlDataAdapter(query, conn);
            DataSet ds = new DataSet();
            sda.Fill(ds, "Customers");

            return ds;
        }

        public void UpdateCustomersDataSet(DataSet ds)
        {
            //string query = "SELECT * FROM Customers";
            //SqlDataAdapter sda = new SqlDataAdapter(query, myConnection);
            //SqlCommandBuilder scb = new SqlCommandBuilder(sda);
            //myConnection.Open();
            //sda.Update(ds, "Customers");
            //myConnection.Close();

            //SqlDataAdapter custDA = new SqlDataAdapter("SELECT CustomerID, CompanyName FROM Customers ORDER BY CustomerID", myConnection);

            //// The Update command checks for optimistic concurrency violations in the WHERE clause.
            //custDA.UpdateCommand = new SqlCommand("UPDATE Customers (CustomerID, CompanyName) VALUES(@CustomerID, @CompanyName) " +
            //                                    "WHERE CustomerID = @oldCustomerID AND CompanyName = @oldCompanyName", myConnection);
            //custDA.UpdateCommand.Parameters.Add("@CustomerID", SqlDbType.NChar, 5, "CustomerID");
            //custDA.UpdateCommand.Parameters.Add("@CompanyName", SqlDbType.NVarChar, 30, "CompanyName");

            //// Pass the original values to the WHERE clause parameters.
            //SqlParameter myParm;
            //myParm = custDA.UpdateCommand.Parameters.Add("@oldCustomerID", SqlDbType.NChar, 5, "CustomerID");
            //myParm.SourceVersion = DataRowVersion.Original;
            //myParm = custDA.UpdateCommand.Parameters.Add("@oldCompanyName", SqlDbType.NVarChar, 30, "CompanyName");
            //myParm.SourceVersion = DataRowVersion.Original;

            //// Add the RowUpdated event handler.
            //custDA.RowUpdated += new SqlRowUpdatedEventHandler(OnRowUpdated);

            ////DataSet custDS = new DataSet();
            ////custDA.Fill(ds, "Customers");

            //// Modify the DataSet contents.
            //custDA.Update(ds, "Customers");

            //foreach (DataRow myRow in ds.Tables["Customers"].Rows)
            //{
            //    if(myRow.HasErrors)
            //        throw new Exception(myRow[0] + "\n" + myRow.RowError);
            //}

            //The Adapter
            SqlDataAdapter custDA = new SqlDataAdapter();

            //The Insert commands
            custDA.InsertCommand = new SqlCommand("INSERT INTO Customers (CustomerID, CompanyName) Values(@CustomerID, @CompanyName)", conn);
            custDA.InsertCommand.Parameters.Add("@CustomerID", SqlDbType.NChar, 5, "CustomerID");
            custDA.InsertCommand.Parameters.Add("@CompanyName", SqlDbType.NChar, 15, "CompanyName");

            //The Update commands
            custDA.UpdateCommand = new SqlCommand("UPDATE Customers Set CustomerID = @CustomerID, CompanyName = @CompanyName WHERE CustomerID = @OldCustomerID", conn);
            custDA.UpdateCommand.Parameters.Add("@CustomerID", SqlDbType.NChar, 5, "CustomerID");
            custDA.UpdateCommand.Parameters.Add("@CompanyName", SqlDbType.NChar, 15, "CompanyName");
            SqlParameter myParm = custDA.UpdateCommand.Parameters.Add("@OldCustomerID", SqlDbType.NChar, 5, "CustomerID");
            myParm.SourceVersion = DataRowVersion.Original;

            //The Delete commands
            custDA.DeleteCommand = new SqlCommand("DELETE FROM Customers WHERE CustomerID = @CustomerID", conn);
            myParm = custDA.DeleteCommand.Parameters.Add("@CustomerID", SqlDbType.NChar, 5, "CustomerID");
            myParm.SourceVersion = DataRowVersion.Original;

            //Do the Update
            custDA.Update(ds, "Customers");
        }

        protected static void OnRowUpdated(object sender, SqlRowUpdatedEventArgs args)
        {
            if (args.RecordsAffected == 0)
            {
                args.Row.RowError = "Optimistic Concurrency Violation Encountered";
                args.Status = UpdateStatus.SkipCurrentRow;
            }
        }


        public Customer GetById(string customerId)
        {
            Customer customer = null;

            string query = String.Format("SELECT customerID, companyName, contactName, contactTitle, address, city, region, postalCode, country, phone, fax FROM Customers WHERE CustomerID = '{0}'", customerId);

            SqlCommand myCommand = new SqlCommand(query, conn);

            conn.Open();

            SqlDataReader myReader = myCommand.ExecuteReader();
            while (myReader.Read())
            {
                customer = new Customer
                (
                    myReader["customerID"].ToString(),
                    myReader["companyName"].ToString(),
                    myReader["contactName"].ToString(),
                    myReader["contactTitle"].ToString(),
                    myReader["address"].ToString(),
                    myReader["city"].ToString(),
                    myReader["region"].ToString(),
                    myReader["postalCode"].ToString(),
                    myReader["country"].ToString(),
                    myReader["phone"].ToString(),
                    myReader["fax"].ToString()
                );
            }

            conn.Close();

            return customer;
        }

        protected virtual SqlParameter[] GetVariablesByEntity(Customer item)
        {
            SqlParameter customerIdParameter = new SqlParameter("@CustomerId", SqlDbType.NChar, 5);
            customerIdParameter.Value = item.CustomerID;
            SqlParameter companyNameParameter = new SqlParameter("@CompanyName", SqlDbType.NVarChar, 40);
            companyNameParameter.Value = item.CompanyName;
            SqlParameter contactNameParameter = new SqlParameter("@ContactName", SqlDbType.NVarChar, 30);
            contactNameParameter.Value = item.ContactName;
            SqlParameter contactTitleParameter = new SqlParameter("@ContactTitle", SqlDbType.NVarChar, 30);
            contactTitleParameter.Value = item.ContactTitle;
            SqlParameter addressParameter = new SqlParameter("@Address", SqlDbType.NVarChar, 60);
            addressParameter.Value = item.Address;
            SqlParameter cityParameter = new SqlParameter("@City", SqlDbType.NVarChar, 15);
            cityParameter.Value = item.City;
            SqlParameter regionParameter = new SqlParameter("@Region", SqlDbType.NVarChar, 15);
            regionParameter.Value = item.Region;
            SqlParameter postalCodeParameter = new SqlParameter("@PostalCode", SqlDbType.NVarChar, 10);
            postalCodeParameter.Value = item.PostalCode;
            SqlParameter countryParameter = new SqlParameter("@Country", SqlDbType.NVarChar, 15);
            countryParameter.Value = item.Country;
            SqlParameter phoneParameter = new SqlParameter("@Phone", SqlDbType.NVarChar, 24);
            phoneParameter.Value = item.Phone;
            SqlParameter faxParameter = new SqlParameter("@Fax", SqlDbType.NVarChar, 24);
            faxParameter.Value = item.Fax;

            return new SqlParameter[]{
                                         customerIdParameter,
                                         companyNameParameter,
                                         contactNameParameter,
                                         contactTitleParameter,
                                         addressParameter,
                                         cityParameter,
                                         regionParameter,
                                         postalCodeParameter,
                                         countryParameter,
                                         phoneParameter,
                                         faxParameter
                                     };
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using ITU.DAL;
using ITU.BusinessEntities;
using System.Data.SqlClient;

namespace ITU.DAL.SqlServer
{
    public class ProductDataProvider : IProductDataProvider
    {
        private static SqlConnection myConnection = new SqlConnection(DataProviderHelper.ConnectionString);

        public Product GetById(string productId)
        {
            Product p = null;

            string query = String.Format("SELECT ProductID, ProductName, SupplierID, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued FROM products WHERE ProductID = '{0}'", productId);
            SqlCommand myCommand = new SqlCommand(query, myConnection);

            myConnection.Open();

            SqlDataReader myReader;
            myReader = myCommand.ExecuteReader();

            try
            {
                while (myReader.Read())
                {
                    p = new Product(
                    myReader["ProductID"].ToString(),
                    myReader["ProductName"].ToString(),
                    myReader["SupplierID"].ToString(),
                    myReader["CategoryID"].ToString(),
                    myReader["QuantityPerUnit"].ToString(),
                    myReader["UnitPrice"].ToString(),
                    myReader["UnitsInStock"].ToString(),
                    myReader["UnitsOnOrder"].ToString(),
                    myReader["ReorderLevel"].ToString(),
                    myReader["Discontinued"].ToString()
                    );
                }
                // always call Close when done reading.
                myReader.Close();
                // Close the connection when done with it.
                myConnection.Close();
            }
            finally
            {
                myConnection.Close();
            }
            return p;
        }


        public IList<Product> GetProducts(string[] productId)
        {
            IList<Product> products = new List<Product>();

            try
            {
                string query = String.Format("SELECT ProductID, ProductName, SupplierID, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued FROM products WHERE ProductID IN ('{0}')", productId);
                SqlCommand myCommand = new SqlCommand(query, myConnection);

                myConnection.Open();

                SqlDataReader myReader;
                myReader = myCommand.ExecuteReader();
                // Always call Read before accessing data.

                while (myReader.Read())
                {
                    products.Add(new Product(
                        myReader["ProductID"].ToString(),
                        myReader["ProductName"].ToString(),
                        myReader["SupplierID"].ToString(),
                        myReader["CategoryID"].ToString(),
                        myReader["QuantityPerUnit"].ToString(),
                        myReader["UnitPrice"].ToString(),
                        myReader["UnitsInStock"].ToString(),
                        myReader["UnitsOnOrder"].ToString(),
                        myReader["ReorderLevel"].ToString(),
                        myReader["Discontinued"].ToString()
                        ));
                }
                // always call Close when done reading.
                myReader.Close();
                // Close the connection when done with it.
                myConnection.Close();
            }
            finally
            {
                myConnection.Close();
            }
            return products;
        }
    }
}
```

ITU.DAL.Hibernate

```
using System;
using ITU.DAL;

namespace ITU.DAL.Hibernate
{
    class DataProviderFactory : IDataProviderFactory
    {

        public ICustomerDataProvider GetCustomerProvider()
        {
            return new CustomerDataProvider();
        }

        public IOrderDataProvider GetOrderProvider()
        {
            return new OrderDataProvider();
        }

        public IProductDataProvider GetProductProvider()
        {
            return new ProductDataProvider();
        }
    }
}

```

```
using System;
using NHibernate;
using NHibernate.Cfg;

namespace ITU.DAL.Hibernate
{
    internal class HibernateHelper
    {
        private static readonly Configuration config;
        private static readonly ISessionFactory sessionFactory;

        static HibernateHelper()
        {
            config = new Configuration();
            config.AddAssembly("ITU.DAL.Hibernate");

            sessionFactory = config.BuildSessionFactory();
        }

        public static Configuration GetConfig()
        {
            return config;
        }

        public static ISessionFactory GetSessionFactory()
        {
            return sessionFactory;
        }
    }
}

```

```
using NHibernate;

namespace ITU.DAL.Hibernate
{
    public abstract class CommonDataProvider
    {

        protected ISession GetNewSession()
        {
            return HibernateHelper.GetSessionFactory().OpenSession();
        }

        protected void EndSession(ISession session)
        {
            session.Flush();
            session.Close();
        }
    }
}

```

```xml
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  <class name="ITU.BusinessEntities.Customer, ITU.BusinessEntities" table="Customers">
    <id name="CustomerID" column="CustomerId" type="String(5)">
      <generator class="assigned" />
    </id>
    <property name="Address" column="Address" type="String(60)" not-null="false" />
    <property name="City" column="City" type="String(15)" not-null="false" />
    <property name="CompanyName" column="CompanyName" type="String(40)" not-null="true" />
    <property name="ContactName" column="ContactName" type="String(30)" not-null="false" />
    <property name="ContactTitle" column="ContactTitle" type="String(30)" not-null="false" />
    <property name="Country" column="Country" type="String(15)" not-null="false" />
    <property name="Fax" column="Fax" type="String(24)" not-null="false" />
    <property name="Phone" column="Phone" type="String(24)" not-null="false" />
    <property name="PostalCode" column="PostalCode" type="String(10)" not-null="false" />
    <property name="Region" column="Region" type="String(15)" not-null="false" />
  </class>
</hibernate-mapping>
```

```csharp
using System;
using ITU.DAL;
using ITU.BusinessEntities;
using NHibernate;
using System.Data;

namespace ITU.DAL.Hibernate
{
    public class CustomerDataProvider : CommonDataProvider, ICustomerDataProvider
    {

        public void Create(Customer c)
        {
            ISession session = GetNewSession();
            session.Save(c);
            EndSession(session);
        }

        public void Update(Customer c)
        {
            ISession session = GetNewSession();
            session.Update(c);
            EndSession(session);
        }

        public void Delete(Customer c)
        {
            ISession session = GetNewSession();
            session.Delete(c);
            EndSession(session);
        }

        public Customer GetById(string customerId)
        {
            ISession session = GetNewSession();
            Customer c = (Customer) session.Get(typeof(Customer), customerId);
            EndSession(session);
            return c;
        }

        public System.Collections.Generic.IList<Customer> GetCustomers()
        {
            ISession session = GetNewSession();
            System.Collections.Generic.IList<Customer> customers = new System.Collections.Generic.List<Customer>();
            System.Collections.IList l = session.CreateCriteria(typeof(Customer)).List();
            System.Collections.IEnumerator e = l.GetEnumerator();
            while(e.MoveNext())
            {
                customers.Add((Customer) e.Current);
            }
            EndSession(session);
            return customers;
        }

        /// <summary>
        /// Programmatically creates a simple dataset with just the CustomerID and returns it
        /// Just so that the WinClient won't crash and to show that it works with Hibernate
        /// </summary>
        /// <returns></returns>
        public System.Data.DataSet GetCustomersDataSet()
        {
            DataTable dt = new DataTable("Customers");
            DataColumn dc = new DataColumn("CustomerID");
            dc.Caption = "CustomerID";
            dc.DataType = System.Type.GetType("System.String");

            dt.Columns.Add(dc);

            foreach (Customer c in GetCustomers())
            {
                DataRow dr = dt.NewRow();
                dr["CustomerID"] = c.CustomerID.ToString();
                dt.Rows.Add(dr);
            }

            DataSet ds = new DataSet();
            ds.Tables.Add(dt);

            return ds;

        }

        public void UpdateCustomersDataSet(System.Data.DataSet ds)
        {
            throw new NotImplementedException("The method or operation is not implemented.");
        }
    }
}

```

```xml
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  <class name="ITU.BusinessEntities.Product, ITU.BusinessEntities" table="Products">
    <id name="ProductID" column="ProductId" type="String(5)">
      <generator class="assigned" />
    </id>
    <property name="ProductName" type="String(20)" not-null="false"></property>
    <property name="SupplierID" type="String(20)" not-null="false"></property>
    <property name="CategoryID" type="String(20)" not-null="false"></property>
    <property name="QuantityPerUnit" type="String(20)" not-null="false"></property>
    <property name="UnitPrice" type="String(20)" not-null="false"></property>
    <property name="UnitsInStock" type="String(20)" not-null="false"></property>
    <property name="UnitsOnOrder" type="String(20)" not-null="false"></property>
    <property name="ReorderLevel" type="String(20)" not-null="false"></property>
    <property name="Discontinued" type="String(20)" not-null="false"></property>

  </class>
</hibernate-mapping>
```

```
using System;
using System.Collections.Generic;
using System.Text;
using ITU.DAL;
using ITU.BusinessEntities;
using NHibernate;

namespace ITU.DAL.Hibernate
{
    class ProductDataProvider : CommonDataProvider, IProductDataProvider
    {

        Product IProductDataProvider.GetById(string productId)
        {
            ISession session = GetNewSession();
            Product p = (Product)session.Get(typeof(Product), productId);
            EndSession(session);
            return p;
        }

        public IList<Product> GetProducts(string[] productId)
        {
            throw new NotImplementedException("The method or operation is not ↙
    implemented.");
        }
    }
}

```

```xml
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  <class name="ITU.BusinessEntities.Order, ITU.BusinessEntities" table="Orders">
    <id name="OrderID" column="OrderID" type="String(5)">
      <generator class="assigned" />
    </id>
    <property name="CustomerID" type="String(20)" not-null="false"></property>
    <property name="EmployeeID" type="String(20)" not-null="false"></property>
    <property name="OrderDate" type="String(20)" not-null="false"></property>
    <property name="RequiredDate" type="String(20)" not-null="false"></property>
    <property name="ShippedDate" type="String(20)" not-null="false"></property>
    <property name="ShipVia" type="String(20)" not-null="false"></property>
    <property name="Freight" type="String(20)" not-null="false"></property>
    <property name="ShipName" type="String(20)" not-null="false"></property>
    <property name="ShipAddress" type="String(20)" not-null="false"></property>
    <property name="ShipCity" type="String(20)" not-null="false"></property>
    <property name="ShipRegion" type="String(20)" not-null="false"></property>
  </class>
</hibernate-mapping>
```

```
using System;
using System.Collections.Generic;
using System.Text;
using ITU.DAL;
using ITU.BusinessEntities;
using NHibernate;

namespace ITU.DAL.Hibernate
{
    class OrderDataProvider : CommonDataProvider, IOrderDataProvider
    {

        public void Create(Order o)
        {
            ISession session = GetNewSession();
            session.Save(o);
            EndSession(session);
        }

        public void Update(Order o)
        {
            ISession session = GetNewSession();
            session.Update(o);
            EndSession(session);
        }

        public void Delete(Order o)
        {
            ISession session = GetNewSession();
            session.Delete(o);
            EndSession(session);
        }

        public Order GetById(string orderId)
        {
            ISession session = GetNewSession();
            Order o = (Order)session.Get(typeof(Order), orderId);
            EndSession(session);
            return o;
        }

        public IList<Order> GetOrders(string customerId)
        {
            ISession session = GetNewSession();
            System.Collections.Generic.IList<Order> orders = new System.Collections.  ↙
    Generic.List<Order>();
            ICriteria criteria = session.CreateCriteria(typeof(Order))
                                    .Add(NHibernate.Expression.Expression.Eq(     ↙
    "CustomerID", customerId));

            System.Collections.IList l = criteria.List();

            System.Collections.IEnumerator e = l.GetEnumerator();
            while (e.MoveNext())
            {
                orders.Add((Order)e.Current);
            }
            EndSession(session);
            return orders;
        }

        public System.Data.DataSet GetOrdersDataSet()
        {
            throw new NotImplementedException("The method or operation is not      ↙
    implemented.");
        }

        public void UpdateOrdersDataSet(System.Data.DataSet ds)
        {
            throw new NotImplementedException("The method or operation is not      ↙
    implemented.");
        }

        public IList<OrderDetail> GetOrderDetails(string orderId)
        {
            ISession session = GetNewSession();
            System.Collections.Generic.IList<OrderDetail> details = new System.Collections.Generic.List<OrderDetail>();
            ICriteria criteria = session.CreateCriteria(typeof(OrderDetail))
                                            .Add(NHibernate.Expression.Expression.Eq("OrderID", orderId));

            //IQuery q = session.CreateSQLQuery("SELECT OrderID, ProductID, UnitPrice, Quantity, Discount FROM \"Order Details\" WHERE OrderID='" + orderId + "'", "OrderDetail", typeof(OrderDetail));
            //System.Collections.IList l = q.List();

            System.Collections.IList l = criteria.List();

            System.Collections.IEnumerator e = l.GetEnumerator();
            while (e.MoveNext())
            {
                details.Add((OrderDetail)e.Current);
            }
            EndSession(session);
            return details;
        }
    }
}
```

```xml
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  <class name="ITU.BusinessEntities.OrderDetail, ITU.BusinessEntities" table="`Order
  Details`">
    <id name="OrderID" column="OrderID" type="String(5)">
      <generator class="assigned" />
    </id>
    <property name="ProductID" type="String(20)" not-null="false"></property>
    <property name="UnitPrice" type="String(20)" not-null="false"></property>
    <property name="Quantity" type="String(20)" not-null="false"></property>
    <property name="Discount" type="String(20)" not-null="false"></property>
  </class>
</hibernate-mapping>
```

ITU.BLL

```csharp
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using ITU.DAL;
using ITU.DAL.Factory;
using ITU.BusinessEntities;

namespace ITU.BLL
{
    public class CustomerManager
    {
        private ICustomerDataProvider dataProvider;

        public CustomerManager()
        {
            dataProvider = ((IDataProviderFactory) new DataAccessProviderFactory()).
    GetCustomerProvider();
        }

        public IList<Customer> GetAllCustomers()
        {
            return dataProvider.GetCustomers();
        }

        public Customer GetById(string customerId)
        {
            if (customerId == null)
                throw new ArgumentNullException("The customer Id cannot be null");
            else if (customerId.Length == 0)
                throw new ArgumentException("The customer Id cannot be blank");
            return dataProvider.GetById(customerId);
        }

        public void Create(Customer c)
        {
            if (c == null)
                throw new ArgumentNullException("item", "The specified customer cannot
     be null");
            else
                dataProvider.Create(c);
        }

        public void Update(Customer c)
        {
            if (c == null)
                throw new ArgumentNullException("item", "The specified customer cannot
     be null");
            else
                dataProvider.Update(c);
        }

        public void Delete(Customer c)
        {
            if (c == null)
                throw new ArgumentNullException("item", "The specified customer cannot
     be null");
            else
                dataProvider.Delete(c);
        }

        public DataSet GetCustomersDataSet()
        {
            return dataProvider.GetCustomersDataSet();
        }

        public void UpdateCustomersDataSet(DataSet ds)
        {
            dataProvider.UpdateCustomersDataSet(ds);
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using ITU.DAL;
using ITU.DAL.Factory;
using ITU.BusinessEntities;

namespace ITU.BLL
{
    public class ProductManager
    {
        private IProductDataProvider dataProvider;

        public ProductManager()
        {
            dataProvider = ((IDataProviderFactory)new DataAccessProviderFactory()).    ↙
    GetProductProvider();
        }

        public Product GetById(string productId)
        {
            return dataProvider.GetById(productId);
        }
    }
}

```

```csharp
using System;
using System.Collections.Generic;
using System.Text;
using ITU.DAL;
using ITU.DAL.Factory;
using ITU.BusinessEntities;

namespace ITU.BLL
{
    public class OrderManager
    {
        private IOrderDataProvider dataProvider;

        public OrderManager()
        {
            dataProvider = ((IDataProviderFactory)new DataAccessProviderFactory()).
    GetOrderProvider();
        }

        public IList<Order> GetCustomerOrders(string customerId)
        {
            return dataProvider.GetOrders(customerId);
        }

        public IList<OrderDetail> GetOrderDetails(string orderId)
        {
            return dataProvider.GetOrderDetails(orderId);
        }
    }
}

```

Website

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage.master.cs" Inherits="masterpages_MasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        &nbsp;<asp:SiteMapPath ID="SiteMapPath1" runat="server" Font-Names="Verdana" Font-Size="0.8em"
            PathSeparator=" : ">
            <PathSeparatorStyle Font-Bold="True" ForeColor="#507CD1" />
            <CurrentNodeStyle ForeColor="#333333" />
            <NodeStyle Font-Bold="True" ForeColor="#284E98" />
            <RootNodeStyle Font-Bold="True" ForeColor="#507CD1" />
        </asp:SiteMapPath>
        <table style="width: 800px; height: 497px">
            <tr>
                <td style="width: 100px; height: 500px" valign="top">
                    <asp:TreeView ID="TreeView1" runat="server" DataSourceID="SiteMapDataSource1" ImageSet="Simple"
                        ShowLines="True">
                        <ParentNodeStyle Font-Bold="False" />
                        <HoverNodeStyle Font-Underline="True" ForeColor="#5555DD" />
                        <SelectedNodeStyle Font-Underline="True" ForeColor="#5555DD" HorizontalPadding="0px"
                            VerticalPadding="0px" />
                        <NodeStyle Font-Names="Tahoma" Font-Size="10pt" ForeColor="Black" HorizontalPadding="0px"
                            NodeSpacing="0px" VerticalPadding="0px" />
                    </asp:TreeView>
                    <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
                </td>
                <td style="width: 100px" valign="top">
                    <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
                    </asp:ContentPlaceHolder>
                </td>
            </tr>
        </table>
    </div>
    </form>
</body>
</html>
```

```xml
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="Default.aspx" title="Menu"  description="Main Menu">
        <siteMapNode url="pages/CustomerList.aspx" title="Customer list"  description= ↙
  "List of all customers" />
    </siteMapNode>
</siteMap>
```

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Collections.Generic;
using System.Data;
using ITU.BusinessEntities;
using ITU.DAL;
using ITU.BLL;
using System.Threading;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    CustomerManager cManager;
    OrderManager oManager;
    ProductManager pManager;

    public Service ()
    {

        //Uncomment the following line if using designed components
        //InitializeComponent();
        cManager = new CustomerManager();
        oManager = new OrderManager();
        pManager = new ProductManager();
    }

    [WebMethod]
    public Product GetProduct(string productId)
    {
        return pManager.GetById(productId);
    }

    [WebMethod]
    public List<OrderDetail> GetOrderDetails(string orderId)
    {
        return (List<OrderDetail>)oManager.GetOrderDetails(orderId);
    }

    [WebMethod]
    public List<Order> GetCustomerOrders(string customerId)
    {
        return (List<Order>)oManager.GetCustomerOrders(customerId);
    }

    [WebMethod]
    public List<Customer> GetAllCustomers()
    {
        Thread tr = new Thread(doSomeThing);
        tr.Start();
        lock (tr)
        {
            Thread.Sleep(1000);
        }
        return (List<Customer>)cManager.GetAllCustomers();
    }

    [WebMethod]
    public void Update(Customer c)
    {
        cManager.Update(c);
    }

    [WebMethod]
    public void Create(Customer c)
    {
        cManager.Create(c);
    }

    [WebMethod]
    public void Delete(Customer c)
    {
        cManager.Delete(c);
    }

    [WebMethod]
    public Customer GetCustomerById(string customerId)
    {
        return cManager.GetById(customerId);
    }


    [WebMethod]
    public DataSet GetCustomersDataSet()
    {
        Thread tr = new Thread(doSomeThing);
        tr.Start();
        lock (tr)
        {
            Thread.Sleep(1000);
        }

        return cManager.GetCustomersDataSet();
    }

    public void doSomeThing()
    {

    }

    [WebMethod]
    public void UpdateCustomersDataSet(DataSet ds)
    {
        cManager.UpdateCustomersDataSet(ds);
    }

}
```

```
<%@ Page Language="C#" MasterPageFile="~/masterpages/MasterPage.master" AutoEventWireup ↙
    ="true" CodeFile="OrderDetails.aspx.cs" Inherits="pages_OrderDetails" Title=         ↙
    "Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
</asp:Content>


```

```
<%@ Page Language="C#" MasterPageFile="~/masterpages/MasterPage.master" AutoEventWireup="true" CodeFile="CustomerOrders.aspx.cs" Inherits="pages_CustomerOrders" Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
    <asp:ObjectDataSource ID="ObjectDataSource1" runat="server" SelectMethod="GetCustomerOrders"
        TypeName="ITU.BLL.OrderManager">
        <SelectParameters>
            <asp:QueryStringParameter Name="customerId" QueryStringField="CustomerID" Type="String" />
        </SelectParameters>
    </asp:ObjectDataSource>
    <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False" CellPadding="4"
        DataSourceID="ObjectDataSource1" Font-Size="Small" ForeColor="#333333" GridLines="None">
        <FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
        <Columns>
            <asp:HyperLinkField DataNavigateUrlFields="OrderID" DataNavigateUrlFormatString="OrderDetails.aspx?OrderID={0}"
                Text="See Order Details" />
            <asp:BoundField DataField="CustomerID" HeaderText="CustomerID" SortExpression="CustomerID" />
            <asp:BoundField DataField="OrderID" HeaderText="OrderID" SortExpression="OrderID" />
            <asp:BoundField DataField="EmployeeID" HeaderText="EmployeeID" SortExpression="EmployeeID" />
            <asp:BoundField DataField="OrderDate" HeaderText="OrderDate" SortExpression="OrderDate" />
            <asp:BoundField DataField="RequiredDate" HeaderText="RequiredDate" SortExpression="RequiredDate" />
            <asp:BoundField DataField="ShipAddress" HeaderText="ShipAddress" SortExpression="ShipAddress" />
            <asp:BoundField DataField="ShipName" HeaderText="ShipName" SortExpression="ShipName" />
            <asp:BoundField DataField="ShippedDate" HeaderText="ShippedDate" SortExpression="ShippedDate" />
            <asp:BoundField DataField="ShipCity" HeaderText="ShipCity" SortExpression="ShipCity" />
            <asp:BoundField DataField="Freight" HeaderText="Freight" SortExpression="Freight" />
            <asp:BoundField DataField="ShipRegion" HeaderText="ShipRegion" SortExpression="ShipRegion" />
            <asp:BoundField DataField="ShipVia" HeaderText="ShipVia" SortExpression="ShipVia" />
        </Columns>
        <RowStyle BackColor="#EFF3FB" />
        <EditRowStyle BackColor="#2461BF" />
        <SelectedRowStyle BackColor="#D1DDF1" Font-Bold="True" ForeColor="#333333" />
        <PagerStyle BackColor="#2461BF" ForeColor="White" HorizontalAlign="Center" />
        <HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
        <AlternatingRowStyle BackColor="White" />
    </asp:GridView>
</asp:Content>
```

```
<%@ Page Language="C#" MasterPageFile="~/masterpages/MasterPage.master" AutoEventWireup="true" CodeFile="CustomerList.aspx.cs" Inherits="pages_CustomerList" Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
    <asp:ObjectDataSource ID="ObjectDataSource1" runat="server" DataObjectTypeName="ITU.BusinessEntities.Customer"
        DeleteMethod="Delete" SelectMethod="GetAllCustomers" TypeName="ITU.BLL.CustomerManager">
    </asp:ObjectDataSource>
    <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False" DataSourceID="ObjectDataSource1" AllowPaging="True" CellPadding="4" Font-Size="Small" ForeColor="#333333" GridLines="None">
        <Columns>
            <asp:HyperLinkField DataNavigateUrlFields="CustomerID" DataNavigateUrlFormatString="CustomerDetail.aspx?CustomerID={0}"
                Text="Edit" />
            <asp:HyperLinkField DataNavigateUrlFields="CustomerID" DataNavigateUrlFormatString="CustomerOrders.aspx?CustomerID={0}"
                Text="See Orders" />
            <asp:BoundField DataField="CustomerID" HeaderText="CustomerID" SortExpression="CustomerID" />
            <asp:BoundField DataField="CompanyName" HeaderText="CompanyName" SortExpression="CompanyName" />
            <asp:BoundField DataField="Address" HeaderText="Address" SortExpression="Address" />
            <asp:BoundField DataField="PostalCode" HeaderText="PostalCode" SortExpression="PostalCode" />
            <asp:BoundField DataField="City" HeaderText="City" SortExpression="City" />
            <asp:BoundField DataField="Country" HeaderText="Country" SortExpression="Country" />
            <asp:BoundField DataField="Phone" HeaderText="PhoneNumber" SortExpression="PhoneNumber" />
            <asp:BoundField DataField="Fax" HeaderText="FaxNumber" SortExpression="FaxNumber" />
            <asp:BoundField DataField="ContactTitle" HeaderText="ContactTitle" SortExpression="ContactTitle" />
            <asp:BoundField DataField="ContactName" HeaderText="ContactName" SortExpression="ContactName" />
            <asp:BoundField DataField="Region" HeaderText="Region" SortExpression="Region" />
        </Columns>
        <FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
        <RowStyle BackColor="#EFF3FB" />
        <EditRowStyle BackColor="#2461BF" />
        <SelectedRowStyle BackColor="#D1DDF1" Font-Bold="True" ForeColor="#333333" />
        <PagerStyle BackColor="#2461BF" ForeColor="White" HorizontalAlign="Center" />
        <HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
        <AlternatingRowStyle BackColor="White" />
    </asp:GridView>
</asp:Content>
```

```
<%@ Page Language="C#" MasterPageFile="~/masterpages/MasterPage.master" AutoEventWireup="true" CodeFile="CustomerDetail.aspx.cs" Inherits="pages_CustomerDetail" Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
    <table id="TABLE1" style="width: 369px; height: 249px">
        <tr>
            <td align="right" style="height: 20px" valign="top" width="150">
                Customer ID:</td>
            <td align="left" style="width: 100px; height: 20px" valign="top">
                <asp:TextBox ID="tbx_CustomerID" runat="server"></asp:TextBox></td>
        </tr>
        <tr>
            <td align="right" style="height: 25px" valign="top" width="150">
                Company Name:</td>
            <td align="left" style="width: 100px; height: 25px" valign="top">
                <asp:TextBox ID="tbx_CompanyName" runat="server"></asp:TextBox></td>
        </tr>
        <tr>
            <td align="right" style="height: 20px" valign="top" width="150">
                Contact Name:</td>
            <td align="left" style="width: 100px; height: 20px" valign="top">
                <asp:TextBox ID="tbx_ContactName" runat="server"></asp:TextBox></td>
        </tr>
        <tr>
            <td align="right" style="height: 20px" valign="top" width="150">
                Address:</td>
            <td align="left" style="width: 100px; height: 20px" valign="top">
                <asp:TextBox ID="tbx_Address" runat="server"></asp:TextBox></td>
        </tr>
        <tr>
            <td align="right" style="height: 20px" valign="top" width="150">
                City:</td>
            <td align="left" style="width: 100px; height: 20px" valign="top">
                <asp:TextBox ID="tbx_City" runat="server"></asp:TextBox></td>
        </tr>
        <tr>
            <td align="right" style="height: 20px" valign="top" width="150">
                Region:</td>
            <td align="left" style="width: 100px; height: 20px" valign="top">
                <asp:TextBox ID="tbx_Region" runat="server"></asp:TextBox></td>
        </tr>
        <tr>
            <td align="right" style="height: 20px" valign="top" width="150">
                Postal Code:</td>
            <td align="left" style="width: 100px; height: 20px" valign="top">
                <asp:TextBox ID="tbx_PostalCode" runat="server"></asp:TextBox></td>
        </tr>
        <tr>
            <td align="right" style="height: 20px" valign="top" width="150">
                Country:</td>
            <td align="left" style="width: 100px; height: 20px" valign="top">
                <asp:TextBox ID="tbx_Country" runat="server"></asp:TextBox></td>
        </tr>
        <tr>
            <td align="right" style="height: 20px" valign="top" width="150">
                Phone:</td>
            <td align="left" style="width: 100px; height: 20px" valign="top">
                <asp:TextBox ID="tbx_Phone" runat="server"></asp:TextBox></td>
        </tr>
        <tr>
            <td align="right" style="height: 20px" valign="top" width="150">
                Fax:</td>
            <td align="left" style="width: 100px; height: 20px" valign="top">
                <asp:TextBox ID="tbx_Fax" runat="server"></asp:TextBox></td>
        </tr>
        <tr>
            <td align="right" colspan="2" style="height: 5px" valign="top" width="150">
                <asp:Button ID="btn_UpdateCustomer" runat="server" Text="Update Customer" OnClick="btn_UpdateCustomer_Click" /></td>
        </tr>
    </table>
</asp:Content>
```

```csharp
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using localhost;

public partial class pages_CustomerDetail : System.Web.UI.Page
{
    localhost.Service wS;
    Customer customer;

    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack && ReqCustomerID != String.Empty)
        {
            wS = new localhost.Service();
            customer = wS.GetCustomerById(ReqCustomerID);
            this.ActiveCustomer = customer;
        }
        //Load the user interface with data for a given customer
        this.DoUiSetup();
    }


    private void DoUiSetup()
    {
        tbx_CustomerID.Text = ActiveCustomer.CustomerID;
        tbx_ContactName.Text = ActiveCustomer.ContactName;
        tbx_Country.Text = ActiveCustomer.Country;
        tbx_Address.Text = ActiveCustomer.Address;
        tbx_City.Text = ActiveCustomer.City;
        tbx_Fax.Text = ActiveCustomer.Fax;
        tbx_Phone.Text = ActiveCustomer.Phone;
        tbx_PostalCode.Text = ActiveCustomer.PostalCode;
        tbx_Region.Text = ActiveCustomer.Region;
    }

    private string ReqCustomerID
    {
        get
        {
            return Request.QueryString["CustomerID"] != null ? Request.QueryString["CustomerID"] : String.Empty;
        }
    }

    private Customer ActiveCustomer
    {
        get
        {
            return (Customer)Session["ThisCustomer"] !=null ? (Customer)Session["ThisCustomer"] : null;
        }
        set
        {
            Session["ThisCustomer"] = value;
        }
    }

    protected void btn_UpdateCustomer_Click(object sender, EventArgs e)
    {
        string customerId = this.ReqCustomerID;
        Customer customer = null;

        if (customerId == string.Empty)
            customer = new Customer();
        else
```

```
            customer = this.ActiveCustomer;

            customer.CustomerID = tbx_CustomerID.Text;
            customer.ContactName = tbx_ContactName.Text;
            customer.Country = tbx_Country.Text;
            customer.Address = tbx_Address.Text;
            customer.City = tbx_City.Text;
            customer.Fax = tbx_Fax.Text;
            customer.Phone = tbx_Phone.Text;
            customer.PostalCode = tbx_PostalCode.Text;
            customer.Region = tbx_Region.Text;

            new localhost.Service().Update(customer);
    }
}
```

ITU.WinClient

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
        <sectionGroup name="userSettings" type="System.Configuration.UserSettingsGroup, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >
            <section name="ITU.DAL.SqlServer.App" type="System.Configuration.ClientSettingsSection, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" allowExeDefinition="MachineToLocalUser" requirePermission="false" />
        </sectionGroup>
    </configSections>
    <userSettings>
        <ITU.DAL.SqlServer.App>
            <setting name="myConnection" serializeAs="String">
                <value>Data Source=(local)\SQLEXPRESS;Initial Catalog=Northwind;Integrated Security=True</value>
            </setting>
        </ITU.DAL.SqlServer.App>
    </userSettings>
</configuration>
```

```csharp
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;


namespace ITU.WinClient
{
    public partial class Customers : Form
    {
        private localhost.Service wS;
        private DataSet ds;
        private localhost.Customer customer;

        public Customers()
        {
            InitializeComponent();
            waitStatus1.Hide();
            wS = new localhost.Service();
        }

        /*
         *      Sync call to webservice, get all the customers in a DataSet.
         *      This will only work in ASP.NET 2.0
         */
        private void btn_GetAllCustomersSync_Click(object sender, EventArgs e)
        {
            waitStatus1.Show();
            waitStatus1.Waiting(sender, e);
            this.UpdateView();
        }


        /*
         *
         *      Async call to webservice, get all the customers in a DataSet.
         *      This will only work in ASP.NET 2.0
         *
         */
        private void btn_GetAllCustomersAsync_Click(object sender, EventArgs e)
        {
            waitStatus1.Show();
            wS.GetCustomersDataSetCompleted += new localhost.GetCustomersDataSetCompletedEventHandler(GetCustomersDataSetCompleted);
            wS.GetCustomersDataSetAsync();
        }

        void GetCustomersDataSetCompleted(Object source, localhost.GetCustomersDataSetCompletedEventArgs e)
        {
            this.UpdateView(e.Result);
        }


        /*
         *
         *      Async call to webservice, update the customers in the DataSet.
         *      This will only work in ASP.NET 2.0
         *
         */
        private void btn_UpdateAllCustomersAsync_Click(object sender, EventArgs e)
        {
            waitStatus1.Show();
            wS.UpdateCustomersDataSetCompleted += new localhost.UpdateCustomersDataSetCompletedEventHandler(UpdateCustomersDataSetCompleted);
            wS.UpdateCustomersDataSetAsync(this.ds);
        }

        void UpdateCustomersDataSetCompleted(object sender, AsyncCompletedEventArgs
e)
        {
            this.btn_GetAllCustomersAsync_Click(sender, e);
        }


        /*
         *
         *      Async call to webservice, get all the customers in a Collection.
         *      This will only work in ASP.NET 2.0
         *
         */
        private void btn_CollectionCustomerAsync_Click(object sender, EventArgs e)
        {
            wS.GetAllCustomersCompleted += new localhost.GetAllCustomersCompletedEventHandler(GetAllCustomersCompleted);
            wS.GetAllCustomersAsync();
        }

        void GetAllCustomersCompleted(object sender, localhost.GetAllCustomersCompletedEventArgs e)
        {
            localhost.Customer[] customers = e.Result;
            this.UpdateView(customers);
        }


        // Update the dataset (synchrounous call)
        private void btn_UpdateAllCustomersSync_Click(object sender, EventArgs e)
        {
            waitStatus1.Show();
            wS.UpdateCustomersDataSet(this.ds);
            this.UpdateView();
        }


        // Use a dataset
        private void UpdateView()
        {
            waitStatus1.Show();
            this.ds = wS.GetCustomersDataSet();
            dataGridView1.DataSource = ds;
            dataGridView1.DataMember = "Customers";
            waitStatus1.Hide();
        }

        // Use a dataset, that is at hand
        private void UpdateView(DataSet ds)
        {
            waitStatus1.Show();
            this.ds = ds;
            dataGridView1.DataSource = ds;
            dataGridView1.DataMember = "Customers";
            waitStatus1.Hide();
        }

        // Use a collection
        private void UpdateView(localhost.Customer[] customers)
        {
            BindingSource bs = new BindingSource();
            BindingList<localhost.Customer> cList = new BindingList<localhost.Customer>();

            foreach (localhost.Customer c in customers)
            {
                cList.Add(c);
            }

            bs.DataSource = cList;
            dataGridView1.DataSource = bs;

            dataGridView1.Columns["CustomerID"].DisplayIndex = 0;
            dataGridView1.Columns["CompanyName"].DisplayIndex = 1;
            dataGridView1.Columns["ContactName"].DisplayIndex = 2;
            dataGridView1.Columns["Address"].DisplayIndex = 3;
            dataGridView1.Columns["City"].DisplayIndex = 4;
            dataGridView1.Columns["Region"].DisplayIndex = 5;
            dataGridView1.Columns["PostalCode"].DisplayIndex = 6;
        }


        /*
         *      This is where the DataGridView showing the Orders get updated
         */
        private void UpdateViewOrders(localhost.Order[] Orders)
        {
            BindingSource bs = new BindingSource();
            BindingList<localhost.Order> oList = new BindingList<localhost.Order>();

            foreach (localhost.Order ord in Orders)
            {
                oList.Add(ord);
            }

            bs.DataSource = oList;
            dataGridView2.DataSource = bs;

            dataGridView2.Columns["OrderID"].DisplayIndex = 0;
            dataGridView2.Columns["CustomerID"].DisplayIndex = 1;
            dataGridView2.Columns["EmployeeID"].DisplayIndex = 2;
            dataGridView2.Columns["OrderDate"].DisplayIndex = 3;
        }


        /*
         *      This is where the DataGridView showing the Order Details get updated
         */
        private void UpdateViewOrderDetails(localhost.OrderDetail[] OrderDetails)
        {
            BindingSource bs = new BindingSource();
            BindingList<localhost.OrderDetail> oList = new BindingList<localhost.OrderDetail>();

            foreach (localhost.OrderDetail ord in OrderDetails)
            {
                oList.Add(ord);
            }

            bs.DataSource = oList;
            dataGridView3.DataSource = bs;
        }


        /*
         *      This is where the DataGridView showing the Product get updated
         */
        private void UpdateViewProduct(localhost.Product p)
        {
            BindingSource bs = new BindingSource();
            BindingList<localhost.Product> pList = new BindingList<localhost.Product>();

            pList.Add(p);

            bs.DataSource = pList;
            dataGridView4.DataSource = bs;
        }


        /*
         *      This is when the Customer is clicked, hence get the Orders
         */
        private void btn_GetCustomerOrders_Click(object sender, EventArgs e)
        {
            this.selectedCustomer_Click(sender, e);
        }


        private void dataGridView1_CellContentClick(object sender, DataGridViewCellEventArgs e)
        {
            this.selectedCustomer_Click(sender, e);
        }


        /*
         *      This is when the Order is clicked, hence get the Order Details
         */
        private void dataGridView2_CellContentClick(object sender, DataGridViewCellEventArgs e)
        {
            this.selectedOrder_Click(sender, e);
        }


        /*
         *      This is when the Order Detail is clicked, hence get the Product
         */
        private void dataGridView3_CellContentClick(object sender, DataGridViewCellEventArgs e)
        {
            this.selectedProduct_Click(sender, e);
        }


        /*
         *
         *      This is where the Orders get loaded
         *
         */
        private void selectedCustomer_Click(object sender, System.EventArgs e)
        {
            Int32 selectedCellCount = dataGridView1.GetCellCount(DataGridViewElementStates.Selected);
            if (selectedCellCount == 1)
            {
                //Hence the Column chosen is CustomerID
                if (dataGridView1.Columns[dataGridView1.SelectedCells[0].ColumnIndex].HeaderText.ToString() == "CustomerID")
                {
                    localhost.Order[] Orders = wS.GetCustomerOrders(dataGridView1.SelectedCells[0].Value.ToString());
                    UpdateViewOrders(Orders);
                }
            }
        }


        /*
         *
         *      This is where the Order Details get loaded
         *
         */
        private void selectedOrder_Click(object sender, System.EventArgs e)
        {

            Int32 selectedCellCount = dataGridView2.GetCellCount(DataGridViewElementStates.Selected);
            if (selectedCellCount == 1)
            {
                //Hence the Column chosen is CustomerID
                if (dataGridView2.Columns[dataGridView2.SelectedCells[0].ColumnIndex].HeaderText.ToString() == "OrderID")
                {
                    localhost.OrderDetail[] OrderDetails = wS.GetOrderDetails(dataGridView2.SelectedCells[0].Value.ToString());
                    UpdateViewOrderDetails(OrderDetails);
                }
            }
        }


        /*
         *
         *      This is where the Product get loaded
         *
         */
        private void selectedProduct_Click(object sender, System.EventArgs e)
        {

            Int32 selectedCellCount = dataGridView3.GetCellCount
    (DataGridViewElementStates.Selected);
            if (selectedCellCount == 1)
            {
                //Hence the Column chosen is CustomerID
                if (dataGridView3.Columns[dataGridView3.SelectedCells[0].ColumnIndex]
    .HeaderText.ToString() == "ProductID")
                {
                    localhost.Product p = wS.GetProduct(dataGridView3.SelectedCells
    [0].Value.ToString());
                    UpdateViewProduct(p);
                }
            }
        }
    }
}
```

```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Text;
using System.Windows.Forms;

namespace ITU.WinClient
{
    public partial class WaitStatus : UserControl
    {
        public WaitStatus()
        {
            InitializeComponent();
        }

        public void Waiting(object sender, EventArgs e)
        {
            this.Visible = true;
        }

        public void NotWaiting(object sender, EventArgs e)
        {
            this.Visible = false;
        }

    }
}

```