HAAGA-HELIA
University of Applied Sciences

Developing an ASP.NET MVC 3 web application using Windows Communication Foundation

Md. Sarwar Jahan

Bachelor's Thesis
BIT
November 2013

HAAGA-HELIA
University of Applied Sciences

Abstract

Date of presentation

Degree programme

Author Md. Sarwar Jahan	**Year of entry** 2009
Title of report Developing an ASP.NET MVC 3 web application using Windows Communication Foundation	**Number of report pages and attachment pages** 46 + 9
Advisor Juhani Välimäki	

The main focus of this thesis is the documentation regarding the building of a pilot web application using the ASP.NET MVC 3 framework that consumes services from a Windows Communication Foundation(WCF) host.

The objective of this thesis was to produce guidelines for developers to familiarize themselves with the ASP.NET MVC 3 framework and WCF.

The thesis explored different aspects of developing a web application including a concise explanation of requirements analysis and high-level architecture. In addition to clarifying technologies like Mercurial, Ninject and IIS, a prototype was implemented using ASP.NET MVC 3 and WCF to guide developers through the basics of the above mentioned technologies. Images for different steps and code blocks were included wherever needed to make the descriptions clearer.

The thesis resulted in guidelines for developers. By following them, it is possible for developers to create a web application using the ASP.NET MVC 3 framework and WCF. The guidelines can also be used to learn the basics or implement features to a web application with the tools and technologies being used.

Keywords
ASP.NET MVC 3, WCF, Mercurial , Ninject, IIS

Table of contents

Terms and abbreviations

Term or Abbreviation	Elaboration
Abstract class	A class that cannot be instantiated or in other words it is not possible to create an object of this class type.
Anonymous method	A C# method without a name.
ASP.NET	A framework developed by Microsoft for web development.
ASP.NET MVC	One of the Microsoft frameworks for web development that follows MVC pattern (see the term MVC).
ASP.NET MVC 3	Third release of the ASP.NET MVC series.
Brand	Different branches of Company X operated in different countries.
Class	A combination of variables of different types, methods, properties and events to make custom data type.
CSS	Also known as Cascading Style Sheets that is used for styling web pages.
Dependency (in the context of this thesis)	An object of a class type that is instantiated inside a different object of another class type.
Dependency injection	A design pattern to make objects loosely coupled by handing off the dependencies rather than initializing directly.
JS	Short form of JavaScript that is generally used for making dynamic web pages.
LINQ	A library in .NET framework to query different types of data objects.
Lambda expression	Advanced way of writing anonymous methods.
Mercurial	An open source tool for version controlling.
MVC	A pattern (in computer science) for developing software that allows developers to create abstractions by keeping different layers completely separate. MVC stands for "Model"-"View"-"Controller".

Ninject	An open source library for .NET framework to establish the "Dependency Injection" design pattern.
Razor (View Engine)	A syntax for generating web pages dynamically where .NET supported programming languages can simultaneously work with mark-up languages (see also the View Engine term).
Repository	A central location, most often in a server, for the original files.
Test Driven Development (TDD)	A software development process where tests are written before the actual implementation.
TortoiseHG	An IDE (Integrated Development Environment) for Mercurial based version controlling.
View Engine	A way of putting markup languages so that it can interact with server side code.
ASP.NET Web Forms	A way of creating web applications in ASP.NET framework by combining HTML (HyperText Markup Language) and server scripts.
Windows Communication Foundation (WCF)	A framework that allows applications to expose and consume services to communicate with other applications.
WCF host	A software component comprised with all required Dlls (Dynamic-link library) to provide services.
WCF client	A software component comprised with all required Dlls (Dynamic-link library) to consume services.

1 Introduction

The thesis topic came into existence from the need of creating a new web application for Company X. The company has different branches called "brands" inside the company in different countries. All of the brands of Company X follow same line of business and offer their services online. Recently, the company decided to make a new web application for a new brand. Concerning the security aspects and separation of different layers of a web application, development team of the company already decided to use ASP.NET MVC 3. Therefore, main focus of the thesis will be on the technical implementation phase of creating a new MVC 3 web application. The application will be working via services provided by a Windows Communication Foundation (WCF) host which is being set up or implemented in another web application that works as a core for all the web applications of the company. Additional technologies like LINQ, Ninject, javaScript, jQuery, Bootstrap CSS framework and Entity Framework used during the implementation, will be explained where needed.

1.1 Goals

The main focus of this thesis is the documentation of building a web application using ASP.NET MVC 3 that consumes services from a WCF host. Starting from basic, this thesis will cover most of the "How to" scenarios to make a web application up and running. Therefore, all the code and supporting images will be included in the project (unless they are marked as private by the owner of the project). The project will also show the standard way of implementing features in terms of software development. Design of the database will also be included as a private attachment.

The thesis will end up producing a blueprint for the developers to get oriented with ASP.NET MVC 3 and WCF. This can also be used to learn the basics of MVC 3 or implement features for a web application using the technologies mentioned earlier.

1.2 Scope of the project

This thesis is based on a web application that belongs to a certain business category. Though businesses from different categories function in different ways, web applications in general share some common features and characteristics. Implementation of common features of a web application like logging customers in and out, will be the scope of this thesis.

As mentioned earlier, the company has other web applications that allow the company to operate its business in different countries. The web application in focus follows the same business idea as other existing web applications of the company. Therefore requirements and specifications for the new web application are not much different than other existing web applications. As a result, requirement and design documentation for the new web application has been simplified substantially.

Company X uses many third party services and also deals with creating and sending invoices which will not be covered in this thesis. In addition, the overall security part of the application and unit testing in MVC 3 will not be explained either.

1.3 Previous research on this topic

Online business is not a new concept anymore. There are many technologies and patterns invented to make robust and user-friendly web applications. A business might need several kinds of services and support from other third parties or might need to provide services to other businesses. ASP.NET MVC from Microsoft is one of the latest technologies to create web applications. WCF is another technology from Microsoft to both consume and provide services. There are quite many researches made separately on both the technologies. A few of them are reviewed below.

One research in HAAGA-HELIA named *Developing an online store for a startup apparel business,* has been recently made mainly focusing on the business side of the topic. The author mentioned very briefly about the technical issues of the ASP.NET MVC 3 framework and other related topics.

Another research called *An Application Using WCF* was carried out focusing on implementing a service host using WCF to provide services. Brief idea about the implementation of how to consume that service was included in the project.

Lastly there is a research of topic *A novel e-commerce web portal: a student led co-creation case within an SME and a University of Applied Sciences* conducted to create a content management system (CMS) for the back-end user. The front-end of the application is responsible to show the data inserted from the back-end. This research also focused mainly on the theoretical aspects of ASP. Net MVC 3 from the business point of view.

2 Theoretical background

After investigating, a collection of areas related to the background of the thesis topic was revealed. ASP.NET MVC and WCF are the main areas of this thesis. This topic also associates with other areas related to web development like application migration, source control and design pattern. The background theory of this thesis therefore will describe all the areas being found during the investigation.

2.1 ASP.NET MVC framework

ASP.NET MVC is a framework for creating robust web applications following the Model View Controller pattern (Galloway, Haack, Wilson & Allen 2011, 1). MVC pattern was first introduced in 1979 as a "Thing-Model-View-Editor" which was later changed to "Model-View-Controller" for the sake of simplicity. The pattern was brought into making web applications concerning the separation of different layers of codes. For example, in MVC pattern the presentation layer is completely separated from the server side code. Though it brings extra complications to the architecture of the software itself, it allows to make the code more scalable and maintainable. Apart from ASP.NET this framework has been adopted by many other technology providers like JAVA. (Galloway et al. 2011, 2.) There are three main components in MVC (Model, View and Controller) architecture and they are kept separated from each other.

Model

Model is a class or group of classes that the application will deal with, defines the business rule of the organization and about how these classes are going to be handled. In ASP.NET it generally mentioned as the Data-Access Layer of the application. (Galloway et al. 2011, 2-3.)

View

View is the layer for displaying markup languages that generally mean the user interface of the application, but server side codes can also be used here if needed (Galloway et al. 2011, 2-3).

Controller

Controller is a class or group of classes that performs the communication between Model and View. It handles the entire application flow and describes the application related logic. This layer is responsible for responding to user input and renders server side code to HTML by handing over the response to a specific view. In ASP.NET this classes normally have "Controller" at the end of their names. (Galloway et al. 2011, 2-3.)

Razor

ASP.NET team from Microsoft introduced Razor view engine with MVC 3, unlike in MVC1 and MVC2. Razor was designed mainly as a view engine syntax keeping only one main focus, which is *code-focused templating for HTML generation*. Being said, the main resposibility of Razor is to render HTML using as less "syntactic craft" as possible. Razor is very easy to use and it is smart enough to understand the transition between server side code and markup language and vise versa when used as a mixture. (Galloway et al. 2011, 5 & 50-51.)

2.2 Application migration

Transferring an application from one environment to another is commonly known as application migration (Rouse, 2012). The transfer can take place due to many reasons. One classic example can be migrating an ASP.NET Web Form based application to MVC 3. Rouse (2012) also stated in her article that, the migration might be a fairly difficult task because of the difference between the old and new environments. Besides, applications are not normally designed concerning the migration ability. That is because an application is generally created for a particular platform following a specific architecture. In addition, other factors like operating system, management tools, server configuration and storage capacity might be completely different than the original environment where the application was created in the first place.

In addition to the ever growing popularity of MVC pattern, Web Forms are still being used in many organzations. The main reason is Web Forms can be used to create powerful and interactive user interfaces in a very short time. But one of the main

problems in Web Forms is that, it is quite difficult to maintain the separation of code from user interaction. Transition between server side code and markup and vice versa is also bit hard to write in Web Forms. Morever, as it does not support the Model-View-Controller pattern, it is quite difficult to carry out test driven development process. Transferring the code behind files of a Web Form application to a Web API controller, it is possible to turn that application into a MVC one. In this way, the Web API takes advantages of the HTTP and performs many low level tasks automatically. This technique skips many traditional concepts of Web Forms like triggering events by controllers, page load, auto generated server side code, view state and so on. (Vogel, 2013.) Considering the limitations of web forms, developers nowadays tend to migrate their applications to ASP.NET MVC because of its light weight architure and possibility of having smooth test driven development.

2.3 Source control

Source control is an ancient method of tracking the changes made to a file or set of files. Individual change, called "Revision", has unique identity number that carries information like timestamp, information about the person who made the change and some additional information. In software development, where a group of people might change one file simultaneously, organising and maintaining revisions has become very important and complex. It is possible to accomplish this task manually by creating copies of the original files and working on the copied versions. In this way, a certain way of naming the modified files is needed which includes lot of discipline and patience from the team members. Possibility of making in mistakes in such a way is very high. (Wikipedia, 2013a.)

However, there are a number of technologies like Git, Mercurial and Subversion to automate the process of source control. Main concept in most of the automated source controlling technologies is to keep the original files in a central storage (in a server) often called as "Repository". Later, files are copied in a local machine and changes are made locally. Coping files from the main "Repository" is generally known as "Cloning". This allows individual to work on a completely separate version of the original file. After changes have been made, those changes are sent to the main "Repository" which

is commonly known as "Push". It is then the responsibility of the source-controlling tool to merge the changes to the original file. Conflicts might occur when a same part of a file is modified by more than one person. Being automated, most of the tools also provide the ability to manage these conflicts and go back and forth to different versions of a file. (Tortoisehg, 2013; Yeates, 2013.)

Mercurial

Mercurial is a free tool for distributed revision control written mainly using Python (a programming language). It supports most of the major operating systems and it's fully extensible. The internal data structure has made mercurial very fast and easy to use even for large size projects. (Mercurial, 2013.)

2.4 Dependency injection

Dependency injection is a software design pattern that solves the negative aspects of tightly coupled structure. Tight coupling is generally considered as an obligation in a software design. Because in tightly coupled structure, one class explicitly knows or depends on another class. In other words, one class gets instantiated inside another class. This increases the possibility to break the code of a class when changes made to the parent or base class. (Galloway et al. 2011, 273.)

Developers generally take two separate but associated steps in order to reduce coupling. First one is creating an abstraction between the two classes using an interface or an abstract class. Second one is creating the dependencies outside of the dependent class. (Galloway et al. 2011, 273.) By injecting the dependencies in an interface or in a setter method or in the constructor of the dependent class, coupling can be loosened between classes (Carr, 2010).

2.5 Technologies and tools for web development

There are a lot of open source and proprietary tools or frameworks available in the market to build organized and robust web applications. Microsoft has also developed many tools and frameworks that support creating software in Microsoft environment.

Different tools or frameworks can reach to a specific goal. It is becoming more and more important for a Microsoft developer nowadays to know the most commonly used Microsoft tools and frameworks in order to work in the real world.

WCF

WCF which stands for Windows Communication Foundation, is a collection of APIs used for building Service Oriented Architecture (SOA). A WCF host provides the services from a network and can serve many WCF clients. WCF clients on the other hand can also consume services from many WCF hosts at the same time. WCF is platform independent, which means it is possible to host or consume Services from any kind of platform. (James, 2010.)

Entity Framework

Entity Framework also known as EF is a set of open source technologies for .NET framework. Often the underlying technologies are referred as ORM (Object-Relational Mapping). It gives a higher level of abstraction to the developers to work with data with less code and without concerning the actual design of the physical database. (Xia, 2012.)

LINQ

Language Integrated Query also known as LINQ, was introduced in Visual Studio 2008 as a part of .NET framework 3.5 in 2007. LINQ is mainly a collection of features that enhances the query capability of .NET supported programming languages like C# and Visual Basic (MSDN, 2013). Using LINQ, querying different types of data source like objects, XML or data-queries has become very easy in .NET framework. LINQ has three types of queries; LINQ to Objects, LINQ to XML and LINQ to SQL. Each of these query types queries a specific type of data source. (Evjen, Hanselman & Rader 2008, 455.)

IIS

Internet Information Services, formerly known as Internet Information Server, is most commonly known as IIS. It is a collection of internet servers available in most of the

Windows operating systems. (Rouse, 2008.) IIS supports almost all the major application protocols like HTTP, HTTPS, FTP and SMTP (Wikipedia, 2013b). Main purpose of IIS is to provide HTML page for a request over a specific protocol. IIS can also handle server side code and dynamically generate response as a form of HTML based on user-inserted data. It also supports low-level interfaces like ISAPI (Internet Server Application Programming Interface) to enhance its functionalities. (Thomas, 2001.)

Like other Microsoft products, IIS has changed and evolved a lot since its first release. IIS 7 came with huge change in the entire architecture. It is revised with modular architecture that helps users to use the modules they want. These modules are called extensions. In IIS 7 it is possible to uninstall an extension that is never going to be used. (Wikipedia, 2013c.)

2.6 Constructive research

The main purpose of the thesis project is to document the "How to" scenarios regarding building a web application using ASP.NET MVC 3 architecture. This involves providing solutions to each aspect of creating and deploying a web application following the most appropriate and standard guideline.

Constructive research focuses on both practical and theoretical relevant problems and provides solutions to those. Therefore, each entity in such research produces solution for a specific problem. When used in software engineering or computer science, this methodology mainly focuses on improving skills and does not bother about explaining the existing problems or solutions. It deals with finding out a practical problem, providing a solution to the problem and demonstrating that the given solution works. (Lassenius, Soininen & Vanhanen 2011, 3-7.)

3 Development planning

Software development is a perplexed and iterative process. Starting from the beginning, it needs proper planning for successful execution. Generally, customers have vague ideas about what the system should do and abstract idea about what they want. As a result, it is very important to pull out the exact requirements from customers' ideas. After extracting the requirements, it is possible for the developers to understand what they are going to do and for the customers to see what they actually wanted. (Rehman & Paul 2003, 1.)

Company X follows the traditional waterfall method for developing software or creating new features to the existing software. Technical documentation phase takes place after implementing and testing a certain feature. The purpose of this thesis is to explain how certain block of code is working and what it is doing. For the sake of the prototype, some of the initial steps like creating a new project using Visual Studio IDE, cloning repositories and putting the newly created project to the repository using ToroiseHG IDE, will be discussed in chapter 5.

This thesis is divided into three main phases. Each phase will end up producing a detailed outline for both stakeholders and developers to understand the whole system.

3.1 Requirements analysis phase

Requirements analysis also known as requirements engineering describes what the product or software is expected to do. It may refer to a completely new software or new features of an existing software. (Rouse, 2007.) From the point of view of this thesis, it is a combination of both. The web application in focus will be a prototype of the new brand of Company X. To perform the main activities, the prototype will be dependent on a WCF host that lies in another web application of Company X. Figure 1 in the next page shows a simplified work flow of the prototype.

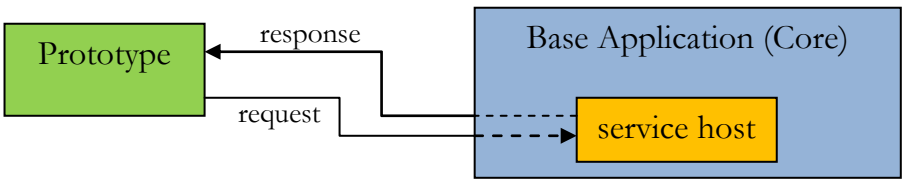


Figure 1. Simplified architecture of the prototype.

Company X has other same kind of web based businesses in different countries. Web applications to operate those businesses are also dependent on the same core. As the company's business idea is same for all the web applications, most of the requirements or features will be the same for the web application in question. Different rules and regulations of the area of opearion have direct effect on this web application. As a result, little modifications to the existing requirements and new features will be made for the new web application. After that, the project manger will make sure with the sponsor's supervisor that all the requirements are appropriate. Finally, the quality of this stage will be assured by the project manager and project supervisor.

3.2 High-level architecture design phase

Main purpose of the software design documentation (SDD) is to guide the involved parties into one common direction by explaining how the requirements should be implemented. (Scott, 2013.) DD can be for both developers and mangers. However, there are some differences between the one for developers with the one for managers. If the document is subjected for the developers then low-level architecture and interaction flow is very important to include. On the other hand, including low-level architecture and interaction flow for managers will only bring complexity to the document. (Hackett, 2007.)

In this thesis one of the very essential parts of the design documentation called Database Design Documentation (DDD) will be explored (Wikipedia, 2013d). Company X keeps its database in a separate server and connects to it through the core. For the sake of simplicity, entities that fulfil the basic needs of a general business will be explored in this thesis (for example customers, addresses and products). The main purpose of this

documentation is to point both the developers and managers to a same source by providing brief idea about the high-level architecture.

3.3 Prototype implementation phase

In terms of software, technical documentation refers to the explanation of the functionalities of that particular software. The main purpose of it is to provide a clear picture to the intended audience about what this particular software is able to do and how it is doing so. In my case, I will be describing most of the major functionalities of Company X's new brand starting from environmental setup to publishing the software online by implementing a prototype.

4 Requirements analysis

Company X deals with certain kind of product (imaginary "P") and has many customers all over the country. Customers are the users of all the web applications of Company X. Company X follows a straight forward development process in order to create the requirements for a brand. See (confidential) attachment 1 to find out company X's development process.

4.1 Requirements from existing system

Requirements of all the web applications of Company X are quite similar as the company X deals with same kind of products in all of them. All the web applications follow quite a simple process, where a user comes to the online site, verifies or registers himself to browse through the products and places an order if he likes it (figure 2).

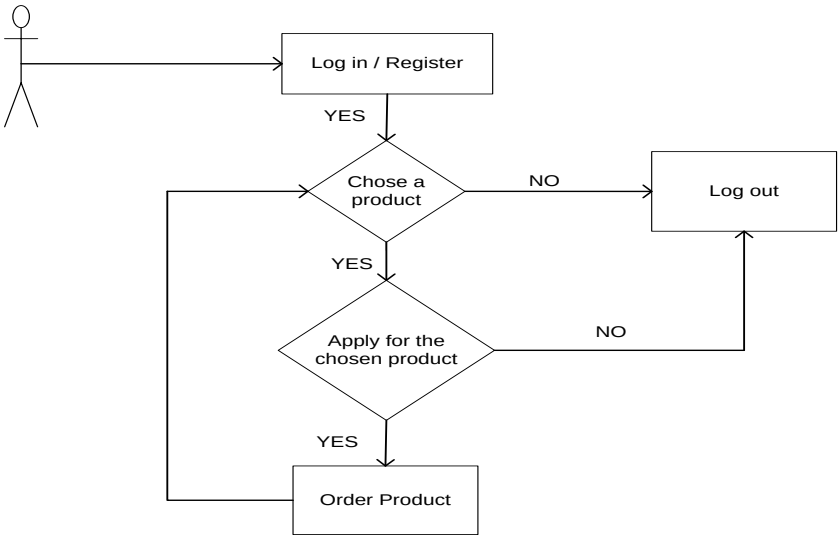


Figure 2. Simplified business process of Company X

However, this process includes many more other processes in between to make the process secure and accurate. See (confidential) attachment 2 to understand Company X's business process in depth.

4.2 Requirements for the new system

There are couple of major differences between the old and new web applications. First difference is that a customer must have an open application in Company X's database in order to order products. A customer is able to open an application when he identifies himself via bank credentials for the first time. If a customer already has an open application, he can then browse through the products by logging himself in using his social security number and password. Each product in the new web application has a price value and all the products belong the open application of a customer (figure 3).

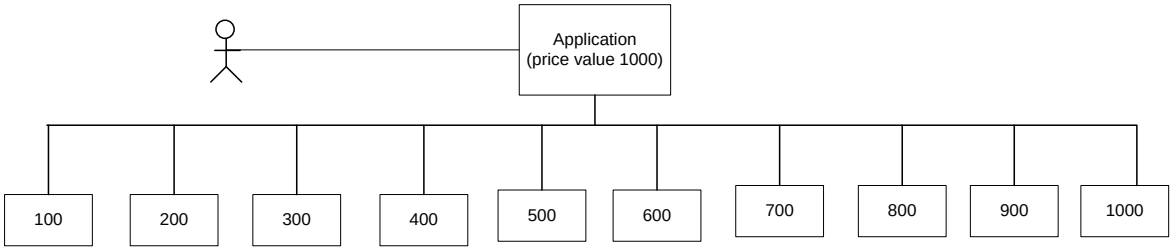


Figure 3. Customer application and product structure

Another difference in the new web application is that the application opened by a customer updates automatically as soon as he pays back the amount of the product he has ordered.

Due to the changed law and need for new features for the new web application led both the requirement engineers and developers to make or modify the existing requirements. However, the process is simplified for the managers and related authorities for better understanding as shown in figure 4 in the next page.

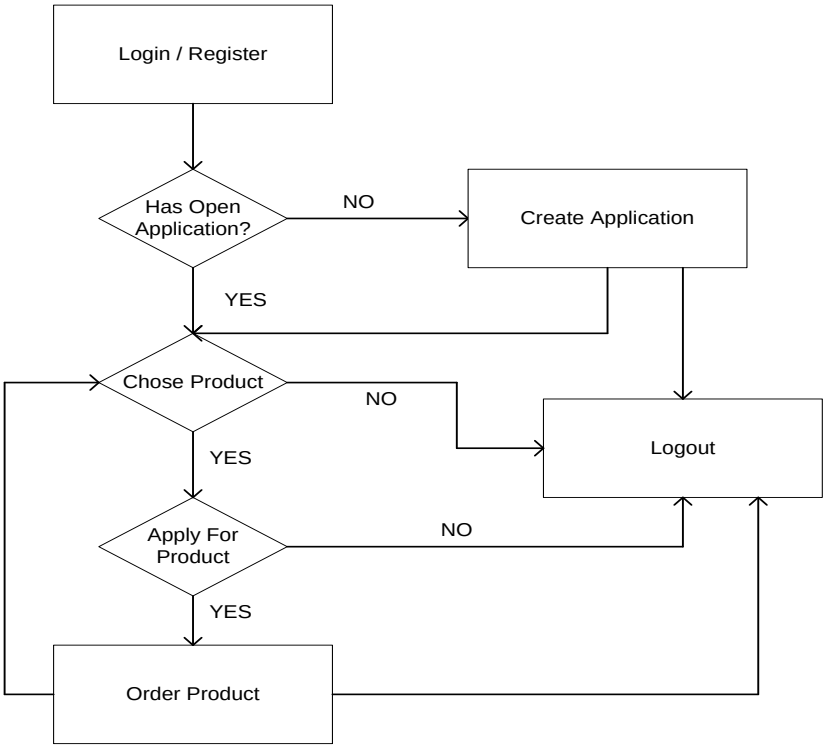


Figure 4. Simplified and revised business process

4.3 Unified requirements

After all the requirements are set, it is the responsibility of all related parties to sit and analyze the whole process. This has many folded benefits. Firstly, it discards all the confusion about the process. Everybody gets a clear picture about what the system is going to do. Through cross checking both engineer and developer can prioritize the requirements and it leads them to form a common ground of understanding. At this point engineers become capable of making high-level design for each requirement.

At this stage, engineers and developers prioritize and split each requirement into different tasks. Developers design low-level architecture and estimate time schedule for each task. After splitting each prioritized requirement into tasks, developers and engineers review the whole business process once again and finalize the requirements.

5 Technical design and implementation

Although there is no physical database involved in the new web application, it is always good to have some knowledge about the high-level architecture of the system and its major data objects. High-level architecture helps developers to understand the work-flow of the application that they are going to build. Exploring different data entities allow developers to understand the navigation and dependencies of each data entity.

5.1 High-level architecture

Company X keeps its database in a separate server and connects to it via the core. Inside the core, Entity Framework is used to make data entities for each table in the database. The new web application will receive all the required data entities by creating a WCF client. WCF client and WCF host communicates and transfers data entities with each other using configurable end-points. An end-point is a set of properties that both the WCF client and host use to make a bridge for mutual handshake. Different services of the WCF host then go through various business rules and finally connect to the database to fetch data. Figure 5 shows the high-level architecture of Company X.

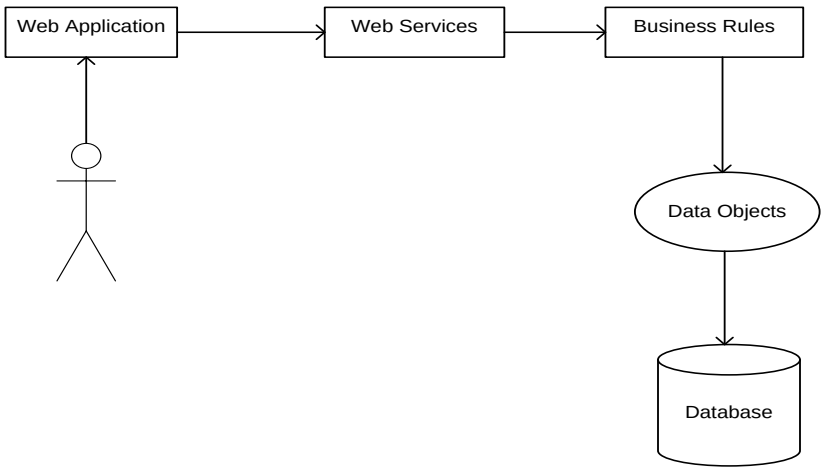


Figure 5. High-level design architecture of Company X

5.2 Major data entities

One of the major entities in Company X's database is the "customers". It stores information related to a customer like "ssn" (social security number), password, registration, marketing and brand. It has a reference named "brand_id" of another entity called "brands" which stores all necessary information associated to a particular brand of Company X. All kind of information related to customers' login is stored in an entity called "customer_logins" where "customer_id" is used as a reference to track customers. All addresses of a customer are stored in "addresses" entity that also has a "customer_id" reference. Attachment (confidential) 3 shows all the fields and relationships of few of the major entities of Company X.

5.3 Prototype implementation

For the demo purpose a prototype is made using ASP.NET MVC 3, as a proof of validity that the procedures undertaken in the following chapters work. While implementing the prototype additional technologies have been used than MVC 3 and WCF as mentioned in chapter 2. There are also some steps that are taken care of both before and after implementing the prototype. Following chapters will also include some of those steps to complete the whole picture in terms of implementing a web application.

5.4 Configuring initial environment

At this point as all the requirements are set and entities are well understood, it is possible to start creating a new web application using Visual Studio IDE. Before that, discussion about setting up Mercurial is needed so that all the files of the web application can be sent to the server for version control. There are other technologies that can be used for the same purpose like Git and SVN. Only Mercurial will be discussed in this thesis since it is being used by Company X for version control. In order to have safe version controlling it is always recommended to create a repository in the server. For simplicity, one of the most popular sites called Bitbucket will be used as dummy server to create a repository.

Creating an account in Bitbucket

Creating an account in Bitbucket is self explanatory and similar to creating an account in any other website. Go to https://bitbucket.org/ and click "Sign up for free" > provide all required information > click "Signup" to create an account.

Creating a repository in Bitbucket

After creating an account in Bitbucket, it is now possible to create a repository there. To do so, click "Repositories" from the main menu > click "Create repositories" > provide necessary information and click "create repository". See attachment 4 for the information need to be provided to create a repository in Bitbucket.

Creating dummy MVC 3 project in Visual Studio 2012

Now, as a repository in a server is created, it is time to create an ASP.NET MVC 3 project using Visual Studio 2012. In Visual Studio 2012 click File > New > Project > Select "ASP.NET MVC 3 Web Application" > fill in all the required information > click ok. See attachment 5 for the required information to create a MVC 3 web application in Visual Studio 2012.

Configuring local repository

After creating the project in Visual Studio and configuring the repository in the server, Now it is possible to store the project in the server. To do so, TortoiseHG needs to be installed first. Go to http://tortoisehg.bitbucket.org/ to download the executable file for windows. Then install it by double clicking the file. Under the hood this will install all the required Mercurial files. When TortoiseHG is installed, right click on the folder where the MVC 3 project was created > click "create repository here" > click ok. This will create a folder named as ".hg" and a file named ".hgignore" in the project folder. The ".hgignore" file is to tell TortoiesHG about which files to ignore while sending the project to the server for the first time. See attachment 6 as a sample "hgignore" file.

Now, right click the project folder again > point to "TortoiseHG" > click "Synchronize", this will bring a window shown in attachment 7. Then go to Bitbucket website > login into the account created earlier using username and password> click to "Over-

view" > click "I have an existing project to push" > copy the URL shown in attachment 8. Paste the copied URL in the "Synchronize" window as shown in the attachment 7 > give it an "alias" name and then > click "save".

Finally, right click to the project folder > click to "Commit". This will bring a window showing that TortoiseHG has tracked all the files that need to send to the server. Notice, files mentioned in the ".hgignore" file are not tracked. They are smoothly ignored by TortoiseHG. Then, write some comment in the text area and select the check box at the top right corner to make all the check boxes checked > click "commit". See attachment 9 for better understanding. Then right click the project folder > click "hg workbench" and in the popped up window click the "push" button as shown in attachment 10. This will send all the files to Bitbucket server.

5.5 Creating service reference

Company X exposes its services via a WCF host. A WCF host is an application that allows creating interoperable services. In a WCF host, classes and interfaces are created by annotating them with .NET specific attributes to define communication contract. On the other hand, a WCF client is responsible to consume the services hosted by a WCF host.

In Visual Studio 2012, under the main project file in solution explorer right click on the folder named "Service Reference" > select "Add Service Reference". In the popped up window type the address of the WCF host > click go. This might take few seconds to find the service host. When the WCF host is found type a name in the namespace text box > click ok (attachment 11). This will auto generate the classes, interfaces and a client along with all the required reference assemblies for the MVC 3 project.

Now this client can be used just by creating an instance of that client class by specifying the given namespace. However, this will give an error while trying to communicate with the WCF host because the WCF host of Company X uses a secured port (:443) to expose its services. Therefore, the WCF client needs to be told explicitly that whenever it is trying to communicate with the WCF host the server certificate validation should

always come back as positive. See figure 6 for the explicit declaration of server certificate validation.

```
public class HomeController : Controller
{
    private readonly [redacted]WCFClient.I[redacted]WCF client = new [redacted]WCFClient.[redacted]WCFClient();

    public ActionResult Index(string message = "")
    {
        System.Net.ServicePointManager.ServerCertificateValidationCallback += (se, cert, chain, sslerror) => true;

        ViewBag.Test = client.TestWCF(); //Put any text here to access it from the View.

        return View();
    }
}
```

Figure 6. Server certificate validation always returns true

The above code snippet means, set the static "ServerCertificateValidationCallback" property of the "ServicePointManager" class to true. The "ServerCertificateValidationCallback" is of type "RemoteCertificateValidationCallback" which is a delegate that takes four arguments and returns a bool (true or false). Therefore, it is possible to create a chain of delegate by calling a method using lambda expression that has a similar signature of taking four arguments and that returns a bool.

5.6 Injecting WCF Client using Ninject

The way a WCF client is created in the previous step is troublesome. The problem is that whenever it needs to be used in the web application, a new instance of it has to be created with explicit instruction for the server certificate validation to return true. This is where Ninject (an open source library for dependency injection for .NET) comes to rescue. Right click on the "References" from the solution explorer > select "Manage NuGet Packages" > Search for "Ninject". From the search result, install "Ninject", "Ninject.MVC 3" and "Ninject.Web.Common" libraries (attachment 12). This will install all the necessary libraries and create a file with ".cs" extension in the "App_Start" folder in the web application's root directory. Rename the file as "IoCConfig.cs" and open it by double clicking. At the bottom of the file, write the following code snippet in the "RegisterServices" method shown in figure 7 to bind the WCF client whenever

the application starts. The code snippet below means whenever the application will ask for a WCF service contract, an instance of the WCF client class will be instantiated.

```
private static void RegisterServices(IKernel kernel)
{
    kernel.Bind<[redacted]WCFClient.I[redacted]WCF>().To<[redacted]WCFClient.[redacted]WCFClient>();
}
```

Figure 7. Setting up Ninject for binding

After that create a class file named "BaseController" inside controller folder by right clicking on it > add > Class. Inherit the class from "Web.MVC.Contoller" and type in the following code snippet shown in figure 8.

```
namespace ThesisProtoMVCApp.Controllers
{
    public class BaseController:Controller
    {
        [Inject]
        public virtual [redacted]WCFClient.I[redacted]WCF client { get; set; }
```

Figure 8. Injecting WCFClient class

Now, open "Global.asax.cs" file from the project root directory and write the server certificate validation code snippet inside the "Application_Start" method as shown in figure 9. This will prevent from writing the server certificate validation code snippet all over in the application.

```
protected void Application_Start()
{

    AreaRegistration.RegisterAllAreas();

    RegisterGlobalFilters(GlobalFilters.Filters);
    RegisterRoutes(RouteTable.Routes);

    System.Net.ServicePointManager.ServerCertificateValidationCallback += (se, cert, chain, sslerror) => true;
}
```

Figure 9. Server certificate validation for the entire application

Finally, it is possible to modify the code in "HomeController" class (shown in figure 6) or in any other class where the application might need to use the WCF client as shown in figure 10. This will work fine because, the "HomeController" class is derived from

the "BaseController" class where the dependency for the WCF client in injected (shown in figure 8).

```
public class HomeController : BaseController
{
    public ActionResult Index(string message = "")
    {
        ViewBag.Test = client.TestWCF(); //Put any text here to access it from the View.

        return View();
    }
}
```

Figure 10. Revised "HomeController" class derived from "BaseController" class

5.7 Creating Models

Models are classes in MVC pattern that can interact with the database, implement business logic and render a view. They make up the domain where the application is going to focus. In ASP.NET MVC 3, models are also used for constructing basic CRUD (create, read, update and delete) scenarios. The process of construction is called "Scaffolding" that generates templates for CRUD functionalities. (Galloway et al. 2011, 69-72.)

In this thesis two major models of Company X will be explored. First one is "LogOnModel", that helps to handle scenarios like logging customer in and logging them out. The second model is called "CustomerDetails" that holds basic information related to a customer like email, phone, and bank account. Creating model is nothing but creating a class in any ASP.NET application. A class generally consists of some public fields, a constructor and required (get and set mainly) properties. In the MVC 3 web application in focus auto generated properties and hidden constructor will be used while creating a class. Hidden constructor means when someone does not not explicitly define a constructor in a class, compiler automatically generates a constructor behind the scene. To create a model, right click on the "Models" Folder in the MVC 3 project > point to "Add" > select "Class". Figure 11 and 12 display both "LogOnModel" and "CustomerDetails" classes (models), that will be used in the prototype.

```
public class LogOnModel
{
    public string ssn { get; set; } //Naming convention was not followed
    public string pin { get; set; } //Please do not break the naming convention
}
```

Figure 11. Model for "Log-on" and "Log-off" functionalities

```
public class CustomerDetails
{
    public int CustomerId { get; set; }
    public string SSN { get; set; }
    public string ForeName { get; set; }
    public string SurName { get; set; }
    public string Street { get; set; }
    public string Zip { get; set; }
    public string City { get; set; }
    public string Country { get; set; }
    public string Phone { get; set; }
    public string Email { get; set; }
    public string BankAccount { get; set; }
    public bool SendMarketingInfoByEmail { get; set; }
    public bool SendMarketingInfoBySMS { get; set; }
    public int InvoiceMethod { get; set; }
    public bool SendInvoiceBy { get; set; }

}
```

Figure 12. Model for "CustomerDetails"

Validation and annotation

Validation is one of the most common features in a web application. Normally validation logics provide information to the users about what are they doing wrong to interact with the web application in their browsers. However, it is also important to have validation in the server side. ASP.NET MVC framework comes with a nice and easy mechanism to handle user inputs. In MVC pattern validation mainly concentrates on validating model values. (Galloway et al. 2011, 69-72.)

There are four basic validation attributes available in ASP.NET MVC in "ComponentModel.DataAnnotation" library. They are "Required", "StringLength", "Range" and "RegularExpression". These attributes are used to validate each property of a model. Figure 13 shows how some of these attributes are used on some the properties of above mentioned "CustomerDetails" model.

```
[Required(ErrorMessage = "Zip Required")]
[RegularExpression("[0-9]{5}", ErrorMessage = "Invalid Zip code")]
public string Zip { get; set; }

[Required(ErrorMessage = "City Required")]
[StringLength(35, ErrorMessage = "Invalid City name", MinimumLength = 2)]
public string City { get; set; }

[Required(ErrorMessage = "Country Required")]
[StringLength(7, ErrorMessage = "Invalid Country name", MinimumLength = 2)]
public string Country { get; set; }

[Required(ErrorMessage = "Phone Required")]
public string Phone { get; set; }

[Required(ErrorMessage = "Email Required")]
[RegularExpression(@"^[a-zA-Z]+(([\'\,\.\- ][a-zA-Z ])?[a-zA-Z]*)*\s+&lt;(\w[-._\w]*\w@\w[-._\w]*\w\.\w{2,3})&gt;$|^(\w[-
                   ErrorMessage = "Invalid E-Mail Address")]
public string Email { get; set; }
```

Figure 13. Using validation attribute

In the "CustomerDetails" model, "StringLength" validation attribute has been used for the "Country" property. It means that the value for "Country" field cannot have less than two or more than seven characters. If a user provides a name that has, less than two or more than seven characters, MVC framework will stop him to proceed while submitting the form. The "Required" validation attribute for the "Country" property simply means a value for the field is mandatory. For the "Email" property, "RegularExpression" validation attribute has been used that forces the user to provide a value for the field, which matches the regular expression.

Of course, it is also possible to make custom validation and there are several ways of implementing it. The simplest approach would be to inherit a class (model) from "IValidatableObject" interface and then implementing that interface. The interface has one simple method that returns a collection of "ValidationResult". In the web application in focus, a user is not allowed to insert any digits for a city name. If there is any digit in the city name, MVC framework will invoke the method for custom validation that will yield a message to the user and stop him proceeding. See figure 14 for the implementation.

```
public class CustomerDetails : IValidatableObject
{
    hide for now

    [Required(ErrorMessage = "City Required")]
    [StringLength(35, ErrorMessage = "Invalid City name", MinimumLength = 2)]
    public string City { get; set; }

    hide for now

    public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
    {
        var client = new [redacted]WCFClient.[redacted]WCFClient();
        var members = new List<string>();
        if (City.Any(char.IsDigit))
        {
            members.Add("City");
            yield return new ValidationResult("City name can not have digit", members);
        }
    }
}
```

Figure 14. Custom validation for the "City" property.

In ASP.NET MVC There are also attributes for displaying data in a certain way. For example, the "pin" property of the "LogOnModel" class practically works as a password. The "DisplayName" attribute can be used here to show a friendly name to the user. There is another attribute called "DataType" that informs the framework to set a specific type for the property while rendering in the browser as an input field. In the example below, the framework will set "password" as a type for the "pin" property while rendering as an input field. However, it is important to keep in mind that, the "DataType" attribute is not an attribute for displaying data. "DataType" is one of the validation attributes that unfortunately does not cause any validation. See figure 15 for the actual code.

```
public class LogOnModel
{
    [Required(ErrorMessage = "SSN is required")]
    [RegularExpression("[0-9]{6}[+-A][0-9]{3}[0-9a-zA-Z]", ErrorMessage = "Wrong SSN")]
    [StringLength(11, ErrorMessage = "{0} SSN must be {2} digit long.", MinimumLength = 11)]
    [Display(Name = "SSN")]
    public string ssn { get; set; }

    [Required(ErrorMessage = "PIN code required")]
    [RegularExpression("[0-9]{4}", ErrorMessage = "Wrong PIN code")]
    [StringLength(4, ErrorMessage = "{0} PIN must be {2} digit long.", MinimumLength = 4)]
    [Display(Name = "PIN")]
    [DataType(DataType.Password)]
    public string pin { get; set; }
}
```

Figure 15. Using display attributes

5.8 Creating Views

View is the final stage of the MVC architecture. It represents the user interface of the web application and allows users to interact with it. First impression and total interaction with the web application begins from the view. In ASP.NET MVC, a "ViewResult" object is handed over to a view by a controller and then the "ViewEngine" renders that object into HTML. Normally a model is accessed by exploring the "ViewData" property from a view in ASP.NET MVC framework. However, in ASP.NET MVC 3 the same data can be accessed by using "ViewBag" property, which is just a wrapper and syntactical sugar of the "ViewData" property. (Galloway et al. 2011, 39-42.)

To create a view open the "HomeController.cs" > right click inside the "Action" called "Index" > click "Add view" > provide a name for the view > leave everything as it is > click "add". This will produce a folder named "Home" inside the "Views" folder and a ".cshtml" file according to the name provided. The ".cshtml" file is the view that will render the model provided by the controller as HTML, when a user will invoke the "Index" action of the "HomeController". See figure 16 as an example. More about "Controller" and "Action" are discussed in the section below.

```
@{
    ViewBag.Title = "Index";
}

<div class="content-box">

    <h2 class="pageTitle">Index</h2>

    <div class="pageContent">

        @if (!string.IsNullOrEmpty(ViewBag.Test))
        {
            <p>@ViewBag.Test</p>
        }

    </div>
</div>
```

Figure 16. Example of a view

The example above shows that the "ViewBag.Test" property is being accessed by the view which was created in the "Index" action of the "HomeController" as shown in figure 17.

```
public class HomeController : BaseController
{
    public ActionResult Index()
    {
        ViewBag.Test = client.TestWCF(); //Put any text here to access it from the View.

        return View();
    }
}
```

Figure 17. Preparing "HomeController" for a view

Strongly-typed view

So far, creating a plain view has been discussed that accessed the built in property of a controller called "ViewBag". However, in a real world scenario a model needs to be passed to a view. One way of passing a model to a view is just assigning the model in the "ViewBag" property inside a controller as shown is figure 18. (Galloway et al. 2011, 43-44.)

```
public ActionResult Index()
{
    var customerId = (int)Session["customer_id"];

    var customer = client.GetCustomerDTOById(customerId);
    var address = client.GetLatestCustomerAddressDTO(customerId);

    var customerModel = new CustomerDetails
    {
        CustomerId = customerId,
        SSN = customer.CustomerSSN,
        Name = address.Forename + " " + address.Surname,
        CustomerBalance = client.GetCustomerBalance(customerId),
    };
    ViewBag.Customer = customerModel;
    return View();
}
```

Figure 18. Passing a model to the view

However, this approach comes with a price. Notice in figure 19 that the "View-Bag.Customer" needed to cast as "IEnumerable<CustomerModel>" before looping through it (Galloway et al. 2011, 43-44).

```
@{
    var customerModel = ViewBag.Customer as IEnumerable<CustomerDetails>;
    if (customerModel != null)
    {
        foreach (CustomerDetails detail in customerModel)
        {
            <p>@detail.Name</p>
            <p>@detail.SSN</p>
        }
    }
}
```

Figure 19. Casting "ViewBag.Customer" as "IEnumerable<CustomerModel>"

There is even a nicer approach to deal with strongly typed model. In this approach, the entire model can be passed to the view by placing it as an argument of the "View()" method. This sets the value of "ViewData.Model" property to the model being passed to the view. Then it is possible to iterate trough the "Model" property of "ViewData" by indicating a fully qualified type name of the model using "@model" declaration. See figure 20 and 21 for details.

```
public ActionResult Index()
{
    var customerId = 0;

    if (Session.Count > 0)
    {
        customerId = (int)Session["customer_id"];
    }
    if (customerId > 0)
    {
        var customer = client.GetCustomerDTOById(customerId);
        var address = client.GetLatestCustomerAddressDTO(customerId);

        var customerModel = new CustomerDetails
        {
            CustomerId = customerId,
            SSN = customer.CustomerSSN,
            Name = address.Forename + " " + address.Surname,
            CustomerBalance = client.GetCustomerBalance(customerId),
        };

        return View(customerModel);
    }

    FormsAuthentication.SignOut();
    return RedirectToAction("Index", "Home");
}
```

Figure 20. Passing strongly typed model to the "View()" method

```
@model ThesisProtoMVCApp.Models.CustomerDetails

@{
    ViewBag.Title = "Index_Temp";
}

<h2>Index_Temp</h2>
<div class="row main">
    <div class="col-lg-6">
        <div class="content-box">
            <h2>Customer</h2>
            <div>
                <p>Welcome, @Html.DisplayFor(model => model.Name)</p>
                <p>Your social security number is: @Html.DisplayFor(model => model.SSN)</p>
                <hr />
                @Html.ActionLink("Show Customer Details", "ShowCustomerDetails", "Customer", new { customerId = Model.CustomerId }, new { @class = "btn btn-primary" })
                @Html.ActionLink("Update Customer Details", "UpdateCustomerDetails", "Customer", new { customerId = Model.CustomerId }, new { @class = "btn btn-primary" })
                <hr />
            </div>
        </div>
        <br/>
        <div class="well">
            <ul class="list-inline">
                <li>Your balance is: @Model.CustomerBalance</li>
                <li class="pull-right">@Html.ActionLink("Log out", "Logout", "Account")</li>
            </ul>
        </div>
    </div>
    <div class="col-lg-6">
        <div class="content-box">
            <h2>Chose your product</h2>
            <div class="well">
                @{Html.RenderAction("ShowProducts", "Product", new { id = Model.CustomerId });}
            </div>
        </div>
    </div>
    <br />
</div>
```

Figure 21. Accessing Model" property of "ViewData" object.

5.9 Creating Controllers

In MVC pattern, a controller performs the leading role by responding to the user input and by modifying the models accordingly. Thus, a controller controls the data flow inside the application by handling both incoming and outgoing data. A method inside controller decides which view to call to hand over the model. (Galloway et al. 2011, 23.) A method in a controller is also known as an "Action".

Creating controller is very simple in ASP.NET MVC 3. To create a controller right click on the "Controllers" folder in the project root directory > point to add > select "Controller" > provide a name for the controller > select "EmptyController" from the template dropdown > click add. This will create a ".cs" file in the "Controllers" folder by the given name. See figure 22 in the next page as a basic controller. This includes one public method (aka Action) named "Index" which returns an "ActionResult" object.

```
public class HomeController : BaseController
{
    public ActionResult Index(string message = "")
    {
        if (!string.IsNullOrEmpty(message))
        {
            TempData["Warning"] = message;
        }
        ViewBag.Test = client.TestWCF(); //Put any text here to access it from the View.

        return View();
    }
}
```

Figure 22. Example of an "Action" in a "Controller"

Unless specified to something else, all the actions in a controller handle "HTTP GET" type requests. In case of "HTTP POST" type requests values from a web form need to be passed to the web application. Therefore, explicit declaration for the action is required so that it can handle "HTTP POST" type requests. This is done by specifying "HttpPost" attribute for the action (see figure 23).

```
[HttpPost, Authorize, ValidateAntiForgeryToken]
public ActionResult UpdateCutomerDetails(CustomerDetails model)
{
    string response;

    if (ModelState.IsValid)
    {
        try
        {
            if (model.SendMarketingInfoByEmail)
            {
                model.InvoiceMethod = (int)InvoiceMethod.EMAIL;
            }
            if (model.SendMarketingInfoBySMS)
            {
                model.InvoiceMethod = (int)InvoiceMethod.PAPER;
            }
            response = Utilities.UpdateCustomerDetails(model);
            TempData["Msg"] = response;
        }
        catch (Exception e)
        {
            //TODO: Use logging here. Dont just eat the exception.
            response = "Could not update customer information";
            TempData["Error"] = response;
        }
    }
    else
    {
        response = "Could not update customer information";
        TempData["Info"] = response;
        return View("UpdateCustomerDetails", model);
    }
    return RedirectToAction("ShowCustomerDetails", "Customer", new { customerId = model.CustomerId });
}
```

Figure 23. "Action" for "HTTP POST" type request

5.10 Configuring routing

Prior to ASP.NET MVC, web pages were served dynamically based on some scripts that are stored in the web server's hard disk. A URL has to point directly to a location in the server for a physical file to invoke some code. However in ASP.NET MVC the routing mechanism performs the same task by building a relationship between the URL and an action residing in a controller. (Galloway et al. 2011, 23.) Figure 24 shows a very basic diagram of the routing mechanism

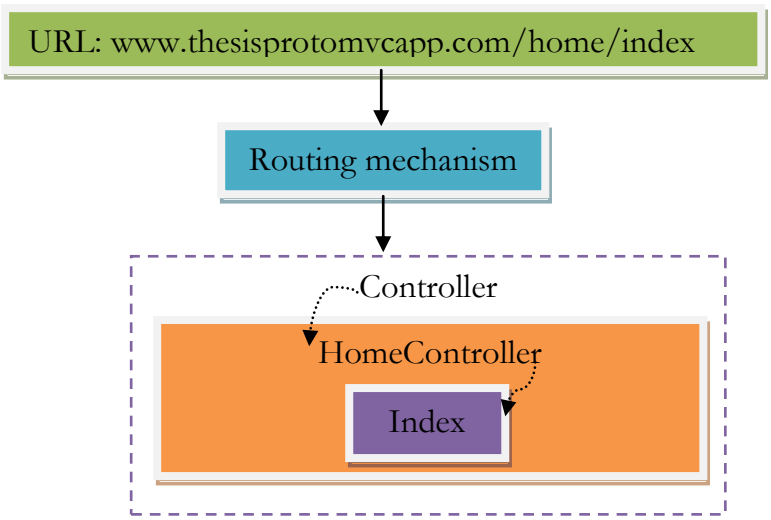


Figure 24. Routing mechanism

ASP.NET MVC uses routing mechanism to match user requests to corresponding action in a controller. The routing engine is given a mapping to follow by the framework as three arguments of the "MapRoute" method. All this happens inside a method called "RegisterRoutes" in "Global.asax.cs" file that is located in the root directory of an ASP.NET MVC 3 web application. Figure 25 shows the default mapping for the routing mechanism given by the framework.

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        "Default", // Route name
        "{controller}/{action}/{id}", // Pattern for the Route to follow
        new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Parameter defaults
    );

}
```

Figure 25. Mapping for the routing mechanism

In the example above the "MapRoute" method is given with three arguments. First argument is the name for the route, which can be anything. Second argument is the pattern for the route to follow. Each section inside the curly braces refers to a specific section in the URL. By comparing the URL used in figure 24 with the pattern used in figure 25, the whole picture gets clearer as shown in figure 26.

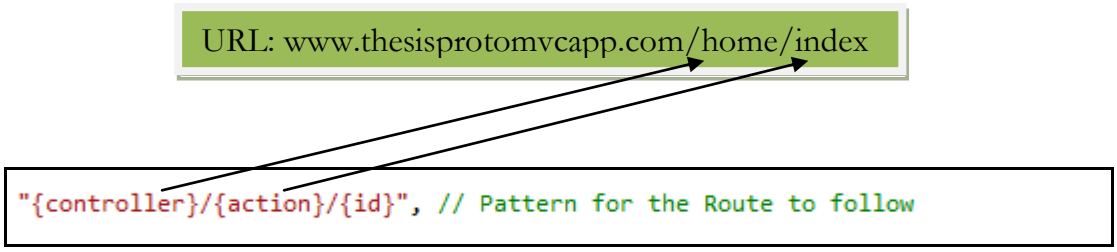


Figure 26. Matched sections of the pattern with an URL

Figure 26 shows that two sections from the route pattern match with two sections in the URL. However, there is no match for the "{id}" section. This is where the third argument of the "MapRoute" method comes into play. As a third argument some default values have been passed to the route by initializing an object, which means if user never specifies anything in the URL the routing mechanism will go to the "HomeCotroller" and invoke the "Index" action. The value for "{id}" section is assigned to "URLParameter.Optional", meaning user does not necessarily have to pass a value for "id". (Galloway et al. 2011, 214-219.)

5.11 Adding Stylesheets and JavaScript

Adding stylesheets (CSS) or client side script (JS) is easy in MVC 3 applications. Nowadays there are plenty of free and ready-to-use libraries for both styling and scripting from different third parties. Developers tend to use these technologies a lot, because these libraries help them to write less code to make a web application more attractive and dynamic. However, using these libraries parallel to own stylesheets and javascripts requires a little bit of understanding about how they work. In the prototype, Bootstrap from the Twitter and jQuery from the jQuery Foundation are used for styling and client side scripting. One thing to keep in mind that, adding CSS files before JS files is an

old practice. In modern browsers, this practice hardly gains any performance (josh3736, 2012). As it cannot be guaranteed that all of the users will have the latest version of browsers, it is safe to follow the old practice and put all the CSS files before the JS files. Another important issue of using "jQuery" libraries is that the main jquery file (either "jquery-1.10.2.min.js" or "jquery-1.10.2.js") must be added before adding any other "JS" file.

In the prototype, to make the "_Layout" file bit cleaner one of the magic folders provided by .NET framework is used and two helper methods are created using "@helper" syntax of Razor. It is worth mentioning here that, in an ASP.NET MVC 3 web application the "_Layout" file works as the "MasterPage" and "@helper" methods are used to make re-useable code blocks. Being mentioned, to implement helper methods, create a folder name "App_Code" by right clicking the solution > point to "Add" > then point to "Add ASP.NET Folder" > select "App_Code". After that, add a ".cshtml" file called "Content" in the "App_Code" folder by right clicking on it and paste the following code snippet inside the "Content.cshtml" file to create helper methods as shown in figure 27.

```
@using System.Web.Mvc
@helper Script(string scriptName, UrlHelper url)
{
    <script src="@url.Content("~/Scripts/" + scriptName)" type="text/javascript"></script>
}
@helper Css(string filename, UrlHelper url)
{
    <link href="@url.Content("~/Content/" + filename)" type="text/css" rel="stylesheet" />
}
```

Figure 27. Creating @helper methods

Then create a partial view named "_Links.cshtml" in the "Shared" folder that is located inside the "Views" folder to use the helper methods created earlier. To do so, right click on the "Shared" folder > point to "add" > select "View...". In the popped up window write "_Links" as a name for the view > check the "Create as a partial view" checkbox > select add. Naming a partial view with a "_" at the beginning of the name is just a convention followed by developers to identify that the "view" is a partial view. In this file, all the required CSS and JS files will be called by providing their names as

an argument of the helper methods as shown in figure 28. Figure 28 indicates that a CSS file called "Site.css" already exists and it is a convention to put all the CSS and image files or folders in the "Content" folder that resides in the root directory. Figure 28 also points that a JS file called "Site.js" has also been created in a folder name "custom". The "custom" folder is then placed inside the "Scripts" folder which also resides in the root directory. The "custom" folder will be holding all the custom JS files.

```
<!--CSS-->
@Content.Css("bootstrap/bootstrap.min.css", Url)
@Content.Css("bootstrap/bootstrap-theme.min.css", Url)
@Content.Css("Site.css", Url)

<!---SCRIPTS-->
@Content.Script("jquery-1.7.1.min.js", Url)
@Content.Script("jquery-ui-1.8.20.js", Url)
@Content.Script("jquery.validate.min.js", Url)
@Content.Script("jquery.validate.unobtrusive.min.js", Url)
@Content.Script("bootstrap/bootstrap.min.js", Url)
@Content.Script("custom/Site.js", Url)
```

Figure 28. Using @helper method inside partial view

Finally, the "_Links" partial view is rendered in the head tag of the "_Layout" file as shown in figure 29.

```
<!DOCTYPE html>
<html>
<head>
    <title>@ViewBag.Title</title>
    <!--LINKS-->
    @Html.Partial("_Links")
</head>
```

Figure 29. Rendering partial view in the "_Layout" file

Now, as all the required CSS and JS files are added it is possible to start writing code for styling and client side scripting. Attachment 13 shows all the CSS codes used for the prototype so far and Figure 30 in the next page shows the script that makes the main navigation bar more interactive.

```
$(document).ready(function () {

    var url = window.location;
    $('ul.main-nav a[href="' + url + '"]').parent().addClass('active');
    $('ul.main-nav').find('a').filter(function () {
        return this.href == url;
    }).parent().toggleClass('active');

    $(document).on('submit', '#LogonForm', function ()...);
});
```

Figure 30. Script for main navigation (see figure 31 for HTML code for the navigation bar)

```
<div class="navbar navbar-inverse">
    <div class="navbar-header">
        <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
        </button>
        @Html.ActionLink("Thesis Prototype", "Index", "Home", null, new { @class = "navbar-brand" })
    </div>
    <div class="navbar-collapse collapse">
        <ul class="nav navbar-nav main-nav">
            <li>@Html.ActionLink("Home", "Index", "Home")</li>
            <li>@Html.ActionLink("Products", "Index", "Product")</li>
            <li class="dropdown">
                <a href="#" class="dropdown-toggle" data-toggle="dropdown">Brands <b class="caret"></b></a>
                <ul class="dropdown-menu">
                    <li><a href="#">Brand A</a></li>
                    <li><a href="#">Brand B</a></li>
                </ul>
            </li>
            <li><a href="#">Contact</a></li>
            <li>@Html.ActionLink("About us", "Index", "About")</li>
        </ul>
    </div>
</div>
```

Figure 31. HTML code for the main navigation bar

5.12 Securing the web application

ASP.NET MVC 3 does not protect a web application automatically. However, most of the common security features come out of the box but protecting the application using those features solely depends on the developers. As the data received from users cannot be trusted, developers should always encode and validate the user inputs, force users to authenticate themselves to the non-public area of the application, should not try to sterilize users HTML inputs manually and should not use accessible cookies via client side scripting. (Galloway et al. 136-137.)

In the prototype, users will be forced to login before they get to see the list of products offered by the company. In MVC 3, this is an easy task to accomplish. All the non-

public actions need to be wrapped with a built-in .NET attribute called "Authorize" wherever users are required to be authenticated beforehand. Figure 32 shows how do achieve this behaviour.

```
public class CustomerController : BaseController
{
    [HttpGet, Authorize]
    public ActionResult Index()
    {
```

Figure 32. Action decorated with "authorize" attribute

Before testing the prototype with the newly added attribute, make sure that some settings are in place. First of all, open the "web.config" file (the one in the root level) > go to the "system.web" section > make sure that authentication mode is set to "Forms". Then check that a valid URL is set for the "forms" section inside the "authentication" section. Figure 33 shows how it should look like.

```
<authentication mode="Forms">
  <forms loginUrl="~/Account/LogOn" timeout="2880" />
</authentication>
```

Figure 33. Valid login URL for authentication

This simply means, redirect the users to the specified "loginUrl" if they are not authenticated yet when accessing a non-public action. Setting a valid URL means, there must be an existing controller and a view matching that URL. Finally make sure that the "HandleErrorAttribute" is added in the "RegisterGlobalFilters" method in "Global.asax" file and the "RegisterGlobalFilters" method is being called in the "Application_Start" method. In case of the prototype, there is a controller name "Account" that has an action inside it called "LogOn" as shown in figure 34.

```
public ActionResult LogOn()
{
    const string warning = "Don't try to be so smart. Login first!";
    return RedirectToAction("Index", "Home",new{message  = warning});
}
```

Figure 34. "LogOn" action inside "AccountController"

Notice that, inside the "LogOn" action there is no view specified for it instead the action has a "RedirectToAction" method as a return value that tells the routing mechanism to go to the "Index" action of the "Home" controller. This is because the view for the "LogOn" action is a "partial view" that is being rendered inside the "_Layout" file which in turn calls the "Index" view of the "Home" controller. View for the "LogOn" action is shown in figure 35.

```
@model ThesisProtoMVCApp.Models.LogOnModel
@if (Request.IsAuthenticated)
{
    Html.RenderAction("RenderLoginInfoBox", "Customer");
}
else
{
    using (Html.BeginForm("LogOnHere", "Account", FormMethod.Post, new { id = "LogonForm", @class = "form-horizontal", @role = "form" }))
    {
    <div class="col-sm-offset-4 col-sm-8">
    </div>
    <div class="form-group">
        <label class="col-sm-4 control-label">
            @Html.LabelFor(m => m.ssn)
        </label>
        <div class="col-sm-8">
            @Html.TextBoxFor(m => m.ssn, new { @class = "form-control", @placeholder = "SSN" })
            <p class="pull-left">@Html.ValidationMessageFor(m => m.ssn)</p>
        </div>
    </div>

    <div class="form-group">
        <label class="col-sm-4 control-label">
            @Html.LabelFor(m => m.pin)
        </label>
        <div class="col-sm-8">
            @Html.PasswordFor(m => m.pin, new { @class = "form-control", @placeholder = "PIN" })
            <p class="pull-left">@Html.ValidationMessageFor(m => m.pin)</p>
        </div>
    </div>

    <div class="form-group">
        <div class="col-sm-offset-4 col-sm-8">
            <input id="submit_button" class="btn btn-success btn-lg btn-block" type="submit" value="Sign in »" />
            <div id="result" class="text-danger hide"></div>
        </div>
    </div>
    }
}
```

Figure 35. View for "LogOn" action (also see figure 36 to see how this view is rendered inside the "_Layout" file)

```
<div class="item-box-content">
    @Html.Partial("_Login", new LogOnModel())
</div>
```

Figure 36. Rendering "_Login" view in the "_Layout"

However, it is worth mentioning that, a membership provider must exist in order to authenticate users. Normally all ASP.NET MVC 3 web applications are configured to a

default membership provider, which can be found in the root level "web.config" file inside the "system.web" section. It is quite easy to set up custom membership provider just by tweaking the existing configuration and some coding. As the prototype gets all customer related information from a WCF host, a separate membership provider was not needed.

5.13 Publishing and Deployment of the web application

As the prototype does not have any database directly involved, deploying will be a fairly easy task. IIS is going to be used as the application server and for deploying the prototype. Therefore, it is important to install IIS in the local machine. To install IIS go to the windows "Start" menu > click "Control Panel" > click "Programs" > click "Turn Windows features on or off" and make sure that the settings in "Internet Information Services" match with figure 37 in the next page.

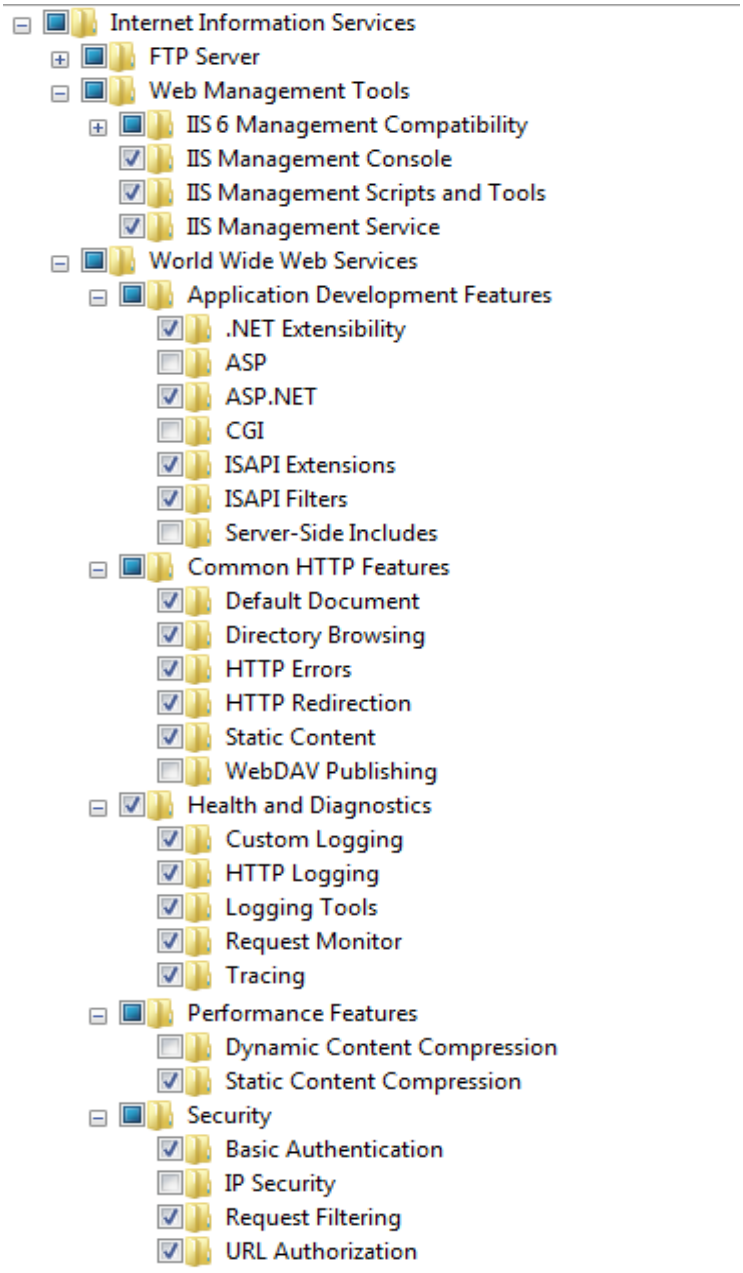


Figure 37. IIS installation options

After installing IIS, make sure that ASP.NET version 4 is installed for IIS, because in the prototype ASP.NET 4.5 framework has been used. To do so, simply open the "command prompt" in administrative mode > navigate to the path shown in figure 38 as the first line and type the command shown as the second line in the same figure.

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319>
C:\Windows\Microsoft.NET\Framework\v4.0.30319>aspnet_regiis.exe -i
```

Figure 38. Installing .NET framework for IIS

After installing ASP.NET 4 for IIS, it is time to create a website inside IIS. Before doing that, the prototype needs to be published to a specific location so that IIS can look for required files to that location. To do so, create a folder in the root level of "C:\\" drive and name it as "Site" and then create another folder inside the "Site" folder and name it as "ThesisProtoMVCApp". Then, in visual studio go to "View" from main tool bar > point to "Toolbars" > select "Web one click publish". This will add the publishing tool in the main toolbar. From the publishing toolbar click the dropdown menu saying "Create Publish Settings" and select "<New...>". In the popped out window select "File System" as publish method and navigate to the "ThesisProtoMVCApp" folder that was created earlier. Click "Settings" from the left side of the window and select "Debug" as configuration and check the checkbox below it saying "Delete existing file prior to publish" and click "Publish".

Finally, open IIS Manager > expand the server by clicking the small arrow at the left corner > right click on the "Sites" folder > select "Add Web Site..." > give the site a name > browse to the folder where the prototype was published from Visual Studio as the "physical path" > give it a "host name" > click ok.

Now, deploying the prototype is completed. It is possible to check if the web application is running or not by selecting the "Sites" in IIS and then by clicking on the "Browse" link from the "Actions" panel.

6 Evaluation and conclusions

The main focus of the thesis was to document the "How to" scenarios of making a web application using ASP.NET MVC 3 using WCF. As a result the requirements and high-level architecture of the web application were narrowed down for all related parties to understand.

In the implementation phase topics unrelated to ASP.NET MVC 3 and WCF were also discussed. This is because, web development is just not about coding rather it includes many other procedures that completes the whole picture in terms of web development.

It is arguable whether the techniques shown in this thesis can be considered as standard or not. In software development, same features can be implemented in many different ways using different tools. Techniques considered as standard by a certain development team or community might not be considered as standard by another development team or community. However, all the techniques and technologies shown in this thesis meet the requirements of the company and are accepted as standard by the development team of Company X.

Lack of knowledge about how to conduct a research was the biggest hinder this entire thesis project. However, at the end of the thesis I learned a lot about conducting research, managing a project and could successfully implement the findings in the project. I also enhanced my knowledge about setting up the environment for version controlling of an ASP.NET MVC 3 web application. Loose coupling and dependency injection have always been tricky chores for me. During the thesis, I got clear idea about implementing a loosely coupled design pattern using "Ninject" framework in an ASP.NET MVC 3 web application.

Reviewing the whole thesis and building a prototype helped me a lot to enhance my knowledge on ASP.NET MVC 3 framework along with several .NET libraries like LINQ. Building a prototype also helped me to improve my C# skills and knowledge about Razor syntax.

Comparing the achievements with the goals, the thesis can be considered as "complete" or "successful". The main objective was to make a blueprint that can act as a guideline for the existing employees of Company X and interested developers on how to build web applications using ASP.NET MVC 3 and WCF. The prototype that has been made by following the techniques mentioned in this thesis work as a benchmark and a proof of validity in this regard.

But in terms of a web application the overall thesis might be considered as "incomplete", because it is missing the low-level architectural design and the complete design of the database. In addition, no information have been provided about the internal architecture of an ASP.NET MVC3 web application like "how routing mechanism selects a controller" and "how a controller knows which action to invoke".

7 Summary

The main goal of the thesis was to document the "How to" scenarios of building a web application using ASP.NET MVC 3 that consumes services provided by a WCF host. Building a web application is not only confined in "coding", steps like version management, understanding the framework in hand, design pattern and security aspects are also needed to be considered. Therefore a prototype is implemented to walk through the readers with the basics of ASP.NET MVC3 and WCF with additional technologies like Mercurial, IIS and Ninject.

Analyzing the requirements of a business and designing the database are also very important in software development. Therefore, necessary information on both have been included in chapter 4 and 5 for better understanding of the overall application from Company X's point of view.

The thesis project is mainly divided into three major parts. First, the requirements of the application were discussed and then brief descriptions about the high-level architecture and some of the major data entities of Company X were provided. Finally, essential steps needed to make an application using ASP.NET MVC 3 framework with external services were described. The final stage also includes guidelines for setting up version control, adding WCF service client, injecting dependencies using "Ninject" framework and deploying the web application using IIS.

7.1 Further development

Due to time and constraint of scope, using Entity framework by involving a physical database was completely skipped in this thesis. In addition, handling different kinds of security threats like XSS (Cross-site scripting) and CSRF (Cross-site request forgery) were not discussed. As a result, any interested party has a huge scope to take this thesis to a next level by analyzing the aspects of using physical database and handling security threats. There is also a scope to dive deep inside the ASP.NET MVC 3 framework and analyze how things work. As mentioned earlier that same sort of implementations can

be done in many ways, different approaches of developing a web application using ASP.NET MVC 3 can also be considered as further development of this thesis.

Bibliography

Bill, E., Scott, H. & Devin R. 2008. Professional ASP.NET 3.5 In C# and VB. Wiley Publishing, Inc., Indianapolis, Indiana.

Carr, R. 2010. Dependency injection.
URL: http://www.blackwasp.co.uk/DependencyInjection.aspx. Accessed: 18.10.2013.

Galloway, J., Haack, P., Wilson, B. & Allen, S. 2011. Professional ASP.NET MVC 3. John Wiley & Sons, Inc. Indianapolis, Indiana.

Hackett, S. 2007. How To Write An Effective Design Document.
URL: http://blog.slickedit.com/2007/05/how-to-write-an-effective-design-document/. Accessed: 19.10.2013.

James, J. 2010. An introduction to Windows Communication Foundation.
URL: http://www.techrepublic.com/blog/software-engineer/an-introduction-to-windows-communication-foundation/. Accessed: 17.10.2013.

Lassenius, C., Soininen, T. & Vanhanen, J. 2001. Constructive Research.
URL:https://www.google.fi/url?sa=t&rct=j&q=&esrc=s&source=web&cd=5&cad=rja&ved=0CFAQFjAE&url=http%3A%2F%2Fwww.soberit.hut.fi%2F~mmantyla%2Fwork%2FResearch_Methods%2FConstructive_Research%2Fconstructive_research.ppt&ei=XIWDUo74M8jStAaOy4HoCQ&usg=AFQjCNESJken323wh5_vy02D9V4uXu2WiQ&sig2=Rh9yowuYzlx_pA7PFyX1kw&bvm=bv.56343320,d.Yms. Accessed: 19.10.2013.

Mercurial. 2013. Mercurial source control management.
URL: http://mercurial.selenic.com/about/. Accessed: 17.10.2013.

Microsoft. 2013. LINQ(Language Integrated Query).
URL: http://msdn.microsoft.com/en-us/library/vstudio/bb397926.aspx. Accessed: 18.10.2013

Rehman, R. & Paul, C. 2003. The Linux Development Platform.
URL: http://www.faqs.org/docs/ldev/0130091154_24.htm. Accessed: 18.10.2013.

Rouse, M. 2012. Application Migration.
URL: http://searchcloudapplications.techtarget.com/definition/application-migration. Accessed 16.10.2013.

Rouse, M. 2008. IIS (Internet Information Server).
URL: http://searchwindowsserver.techtarget.com/definition/IIS. Accessed: 17.10.2013.

Rouse, M. 2007. Requirements analysis (requirements engineering).
URL: http://searchsoftwarequality.techtarget.com/definition/requirements-analysis.
Accessed: 18.10.2013.

Stackoverflow. 2012. Is the recommendation to include CSS before JavaScript invalid?
URL: http://stackoverflow.com/questions/9271276/is-the-recommendation-to-include-css-before-javascript-invalid. Accessed: 20.10.2013

Scott, AC. 2013. How to Write a Software Design Document.
URL: http://www.ehow.com/how_6734245_write-software-design-document.html.
Accessed: 18.10.2013.

Tortoisehg. 2013. Welcome to TortoiseHg's documentation!
URL: http://tortoisehg.bitbucket.org/manual/2.10/. Accessed: 17.10.2013

Thomas, H. 2001. What IIS actually is, tools to use and reference sites.
URL: http://searchwindowsserver.techtarget.com/tip/What-IIS-actually-is-tools-to-use-and-reference-sites. Accessed: 17.10.2013.

Vogel, P. 2013. Migrating ASP.NET Web Forms to the MVC Pattern with the ASP.NET Web API.
URL: http://msdn.microsoft.com/en-us/magazine/jj991978.aspx. Accessed 16.10.2013.

Wikipedia. 2013a. Revision Control.
URL: http://en.wikipedia.org/wiki/Revision_control. Accessed: 16.10.2013.

Wikipedia. 2013b & 2013c. Internet Information Service.
URL: http://en.wikipedia.org/wiki/Internet_Information_Services. Accessed: 17.10.2013.

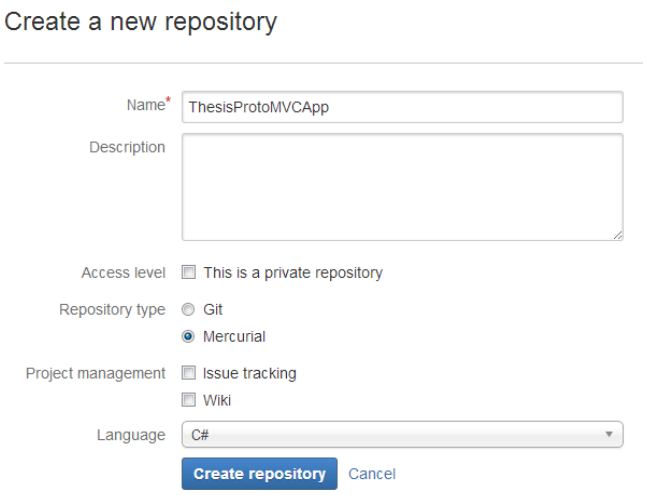
Wikipedia. 2013d. Software documentation.
URL: http://en.wikipedia.org/wiki/Software_documentation. Accessed: 18.10.2013.

Xia, A. 2012. Introduction to Entity Framework.
URL: http://www.codeproject.com/Articles/498885/Introduction-to-Entity-Framework. Accessed: 19.10.2013.

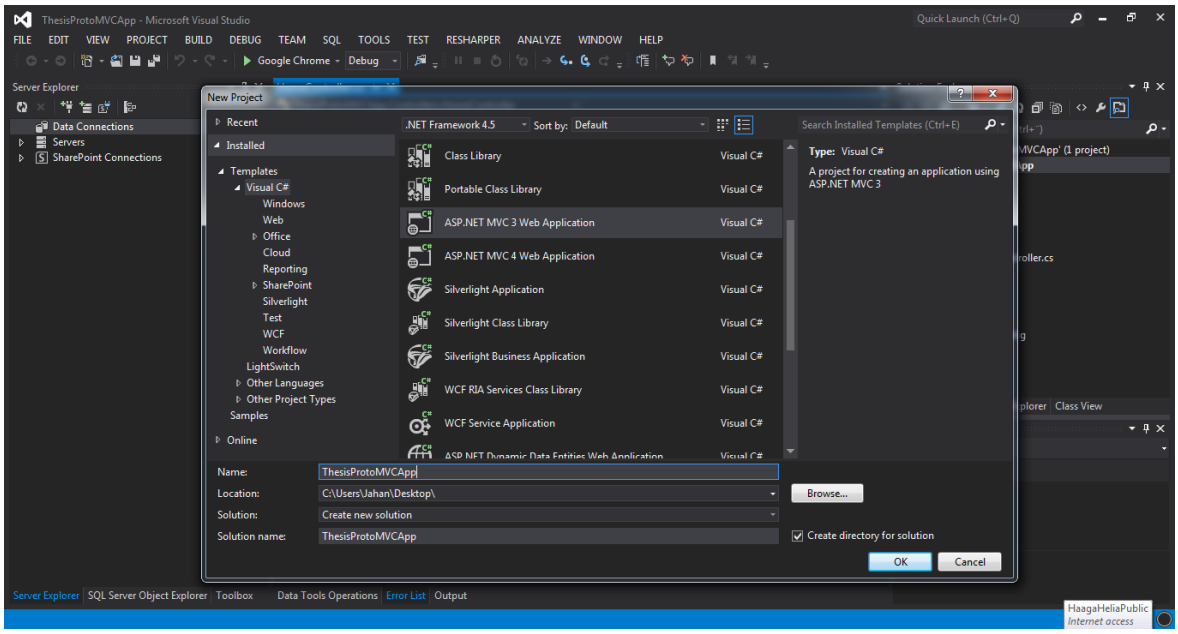
Yeates, S. 2013. What Is Version Control? Why Is It Important For Due Diligence?
URL: http://oss-watch.ac.uk/resources/versioncontrol. Accessed: 16.10.2013

Attachments

Attachment 4: Create repository in Bitbucket



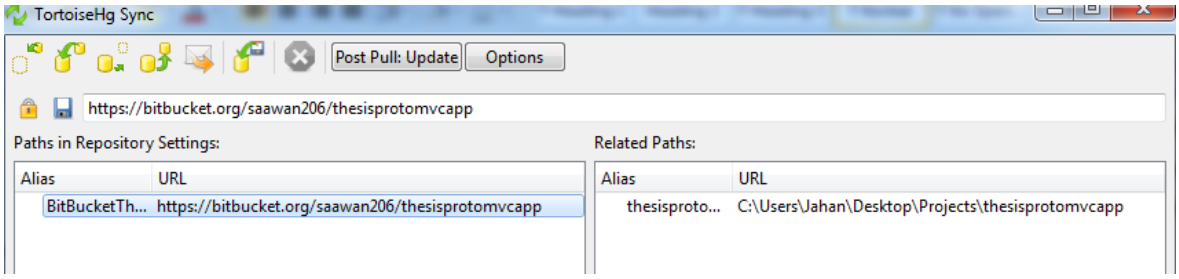
Attachment 5: Create MVC 3 app in Visual Studio 2012



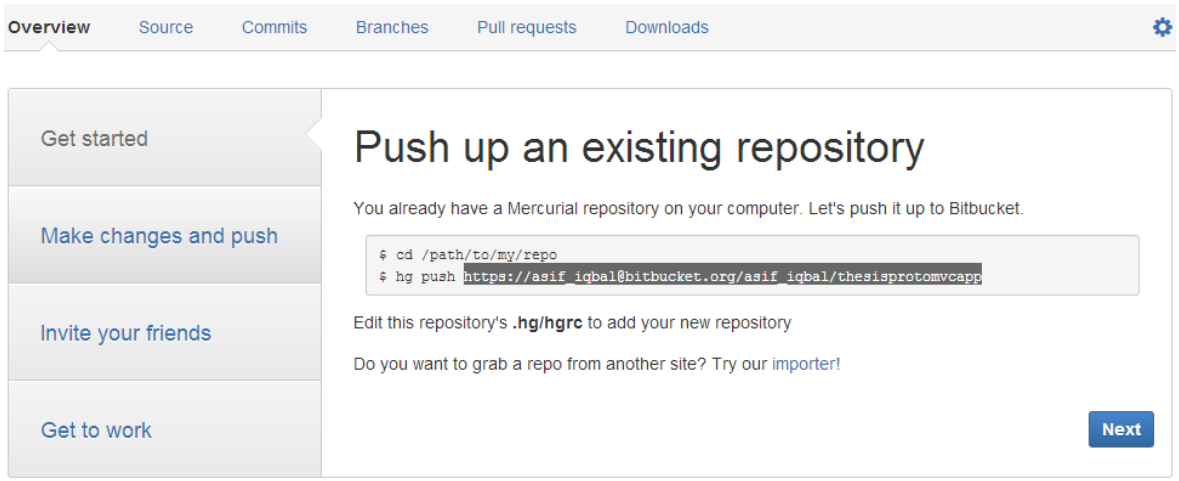
Attachment 6: Sample .hgignore file

```
##This is how you comment out something in this file
syntax: glob
*hg_ignore*

*.orig
*.bak
*.log
*.user
*.*.user
*.cache
*.swp
*.Cache
*.xsd
*.wsdl
*.disco
*.svcinfo
*.MDF
*.ldf

##*.suo
##*.sln
*.sqlsuo
*DotSettings.user
*proj.*
*.Publish.xml

logs/*

**/bin/
**/Bin/
**/obj/

_ReSharper*/
*.ReSharper
*ReSharper.user
*.svn-base
**/.svn/
TestResults*/
.hgtags
```

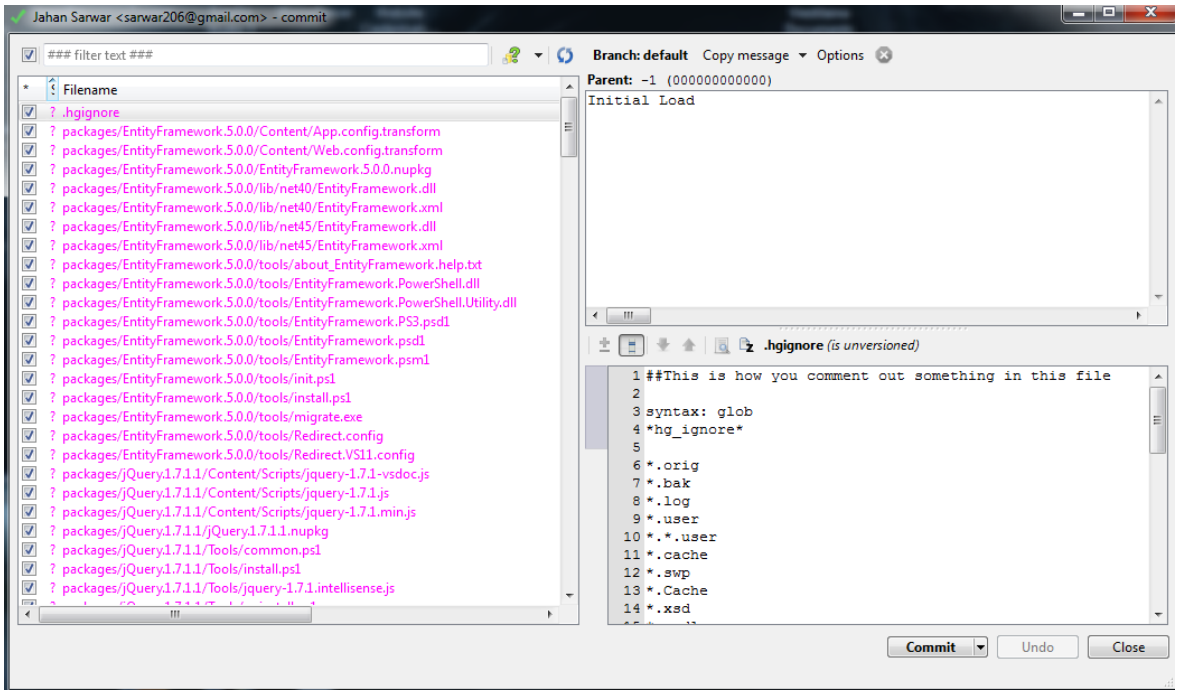
Attachment 7: TortoiseHG synchronize window



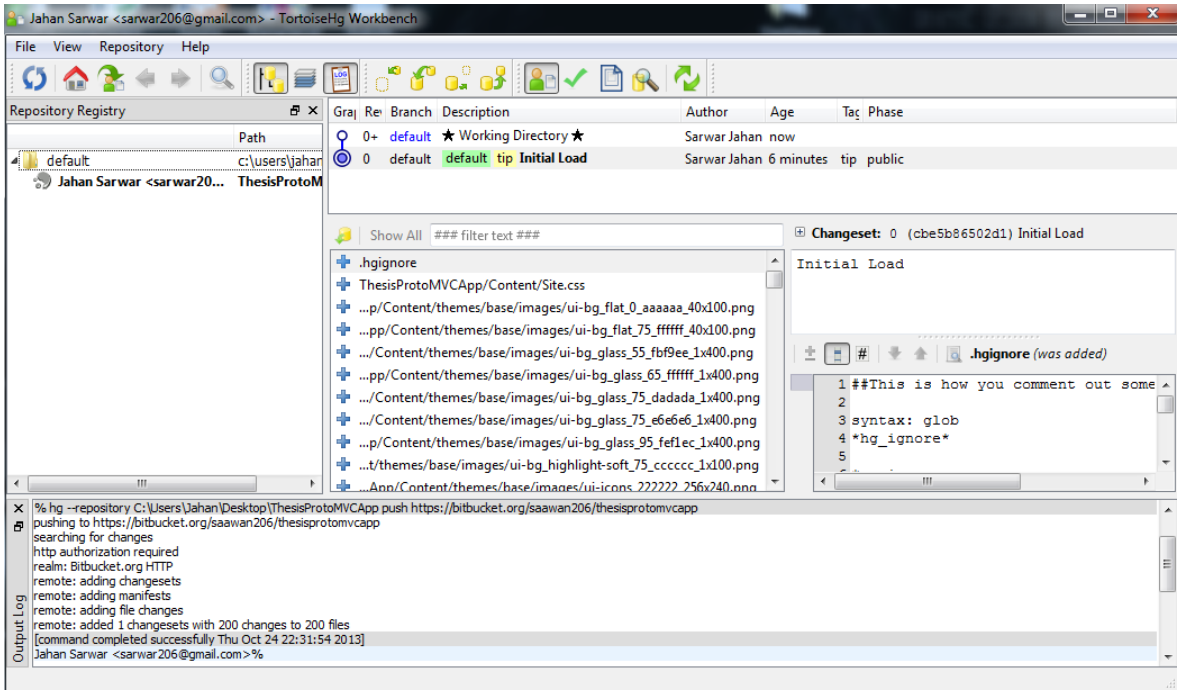
Attachment 8: Copy URL from Bitbucket



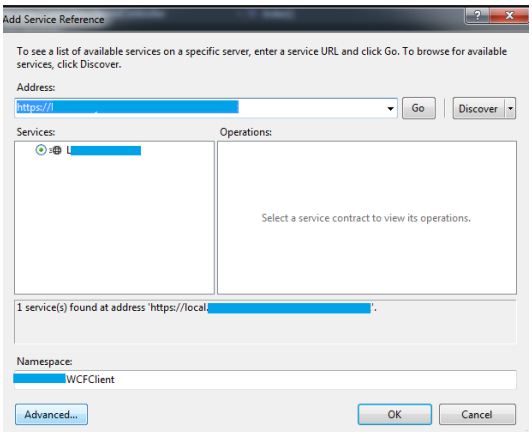
Attachment 9: Tortoise Commit window



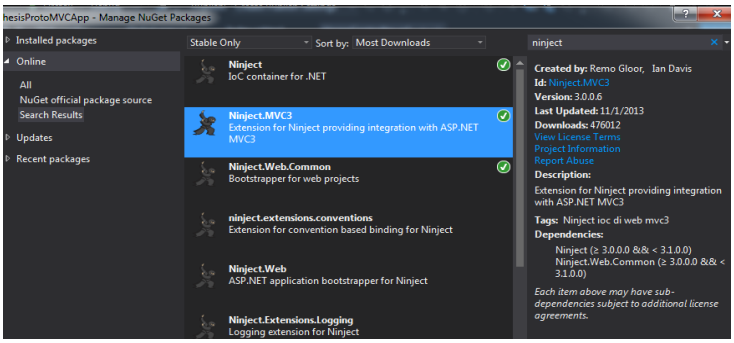
Attachment 10: Tortoise workbench window



Attachment 11: Add service reference



Attachment 12: Installing Ninject libraries



Attachment 13: CSS codes for the project

```
body {background-color: #E5E5E5;border-top-style: solid;border-top-width: 15px;
border-top-color: #333333;}

html, body {height: 100%;}

#wrap {min-height: 100%;height: auto !important;height: 100%;margin: 0 auto -150px;}

.navbar {margin-top: 5px !important;}

.slide {margin-top: -15px;}

.boxes {margin-top: -35px;}

.alert {margin-bottom: 40px !important;}

#push, #footer {height: 150px;}

.item-box h2.item-box-title {background-color: #8EC447;margin-bottom: 0;padding: 5px
0 5px 10px;}

.item-box div.item-box-content {background-color: #96D3D4;padding: 5px 10px 5px
0px;}
```

```
.item-box div.item-box-content #LogonForm {margin-top: 20px;margin-left: -95px;}

.main {margin-top: -55px;}

.content-box h2 {background-color: #96D3D4;margin-bottom: 0;padding: 5px 0 5px
10px;}

.content-box div {background-color: #8EC447;padding: 5px 10px 5px 10px;min-height:
200px;}


#footer {color: #ffffff;background-image: linear-gradient(to bottom, #3C3C3C 0px,
#222222 100%);background-repeat: repeat-x;background-color: #222222;}

.footer-divider {border-right: 1px;border-right-style: inset;border-right-color:
#ffffff;height: 150px;opacity: 0.15;}

#footer .col-lg-4:last-child {opacity: 0.15;}

/* Styles for validation helpers
-----------------------------------------------------------*/
.field-validation-error{color: #ff0000;}

.field-validation-valid{display: none;}

.input-validation-error{border: 1px solid #ff0000;background-color: #ffeeee;}

.validation-summary-errors{font-weight: bold;color: #ff0000;}

.validation-summary-valid{ display: none;}
```