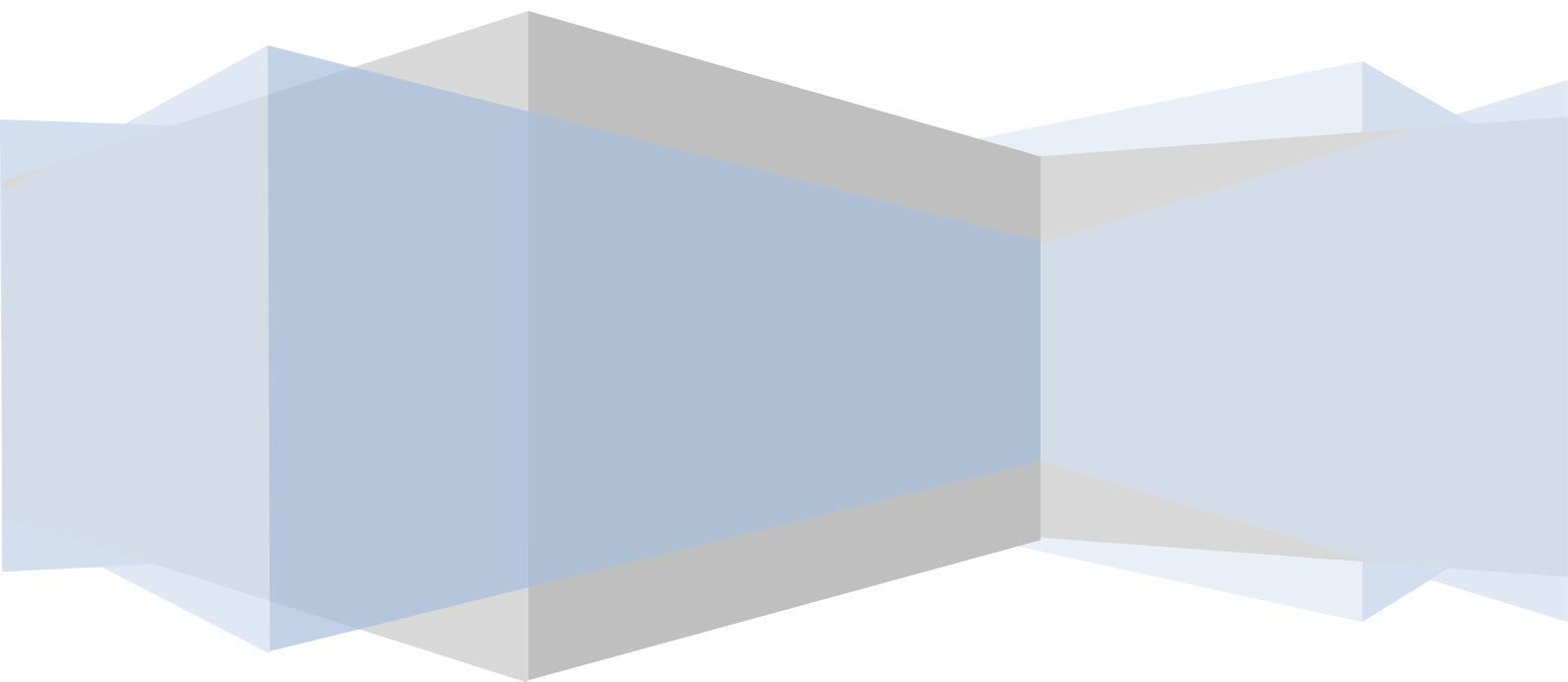


Enterprise .NET Portfolio

Semester 1, 2011

Daniel Lo Nigro – Student ID 6508812



Contents

The contents of my portfolio are in the order shown here:

1. Introduction and Learning summary report
2. Experience report (and associated code)
3. Research report – Security (and associated code)
4. Short report – Security concerns for web services
5. Short report – ASP.NET Web forms
6. Short report – Model-View-Controller architectural pattern
7. Short report – Pros and Cons of the ADO.NET Provider Model
8. Short report – Windows Services
9. Other miscellaneous portfolio writeups, and assignments
10. Assignment code
11. Spikes (and associated code)

Learning Summary Report

1 Introduction

Enterprise .NET is an introduction into the world of .NET technologies used in enterprise environments. Especially with newer .NET versions (3.5 and 4.0), Microsoft have added a wide range of new technologies that enhance the development experience. These include new data access libraries such as Entity Framework, support for new architectural patterns (like ASP.NET MVC, an implementation of the MVC pattern), user interface shininess (WCF and Silverlight), and other overall awesomeness (like LINQ). In my opinion, creating applications with the .NET Framework is fun and efficient.

In my portfolio, I'll demonstrate a number of these technologies, both new and old. I will demonstrate my knowledge of .NET technologies, design patterns and security issues, and how this knowledge applies to all the intended learning outcomes for Enterprise .NET.

2 Reflection

I have previously done a number of .NET subjects at Swinburne, including Object-Oriented Programming and Advanced .NET (both lectured by Andrew Cain), and Database Programming (which I believe was lectured by Clinton Woodward). All three of these subjects had specific goals and intended learning outcomes related to these:

- Object-oriented programming was, as the name suggests, an introduction to object orientation. There was some discussion about .NET-specific functionality, but most of the work was not really C#- or .NET-specific, instead focusing on object-orientation principles (which are language-independent).
- Advanced .NET focused mainly on concurrency constructs in .NET, using C#. This included things like multithreading and the associated problems (like locking). Again, a lot of this was language-independent principles instead of .NET-specific things.
- Database Programming was mainly focused on .NET technologies used to access databases. This included ADO.NET (which is also covered in this portfolio) and NHibernate, as well as design patterns for creating business layers. This was more into the realm of .NET-specific technologies, but didn't cover too much (apart from database technologies, of course!)

Enterprise .NET is the first subject I've done that's been entirely focused on .NET technologies. It was a great unit (one of the best I've done at Swinburne) and the lectures were all very enjoyable. In Enterprise .NET, I have learnt about various .NET technologies, such as COM+, MSMQ, Entity Framework, WCF, the MSDTC, configuration file encryption, Silverlight, and many more. In addition to this, I've also learned about software development patterns (such as patterns used to develop the business layer of an application) and security concerns associated with web-accessible applications.

As I currently work at a company that uses Microsoft products and technologies extensively, the Enterprise .NET unit has helped me significantly. I really do like C# (it is one of my favourite programming languages), so knowing about the technologies used in the industry will definitely help

me both now and in the future. I'm looking forward to using a lot of the technologies I've learnt about in "real" situations. I'm very tempted to rewrite my website using ASP.NET MVC (it currently uses PHP).

3 Self-assessment

This subject has four learning outcomes, which I believe I have successfully addressed to the following levels:

	Adequate	Good	Outstanding	Exemplary
ILO 1: Implement software solutions that illustrate the purpose of .NET technologies related to enterprise application development.	✓	✓	✓	
ILO 2: Describe an architecture for a software solution to a given business scenario, and select and justify the .NET technologies needed for its implementation.	✓			
ILO 3: Describe architectural patterns and best practices for developing enterprise applications and relate these to the .NET technologies covered.	✓	✓		
ILO 4: Explain security issues that arise when developing distributed and service based applications and how these security concerns can be addressed to mitigate potential threats.	✓	✓	✓	✓

3.1 Intended Learning Outcome 1

ILO1: Implement software solutions that illustrate the purpose of .NET technologies related to enterprise application development

I believe that I have addressed this learning outcome to an **outstanding** level. I have written a detailed description on many of the technologies covered in this unit, including a writeup on Windows Forms, and Windows services. In addition to this, I have written a short report on **both** the ADO.NET Provider Model, and ASP.NET Web Forms.

In the experience report about the program I developed, I also wrote about all the .NET technologies chosen for the program, and why they were chosen.

Most of the pieces required for an adequate result were done via the weekly assignment work. The following were done as additional pieces:

- DAL using ADO.NET – Done via the transaction scripts spike
- Web user interface – Done via my main portfolio piece (jobs site)
- MSDTC for distributed transactions – Done via WCF distributed transactions spike
- Client-side rich user interface – Done via password hashing example GUI example

I believe that the experience report, combined with the two short reports and other writeups, is more than sufficient to obtain an "Outstanding" result for this Intended Learning Outcome.

3.2 Intended Learning Outcome 2

ILO 2: Describe an architecture for a software solution to a given business scenario, and select and justify the .NET technologies needed for its implementation.

I believe this is covered in my experience report, to an **adequate** level. The experience report for my portfolio piece describes the business scenario, and gives details of the architecture, including all the .NET technologies used, and why they were chosen. The amount of detail in the experience report should be sufficient to address this intended learning outcome to at least an adequate level.

3.3 Intended Learning Outcome 3

ILO 3: Describe architectural patterns and best practices for developing enterprise applications and relate these to the .NET technologies covered.

I believe this intended learning outcome is covered to a **good** level via the short report I have written on the MVC architectural pattern and its usage in the development of web applications. The MVC pattern was touched on briefly during the UI lectures and hence I believe it counts as an “architectural pattern presented in this unit”.

The adequate pieces are met by the following:

- Business logic implemented using a domain model is shown via the program I developed as part of my portfolio.
- Business logic implemented using transaction scripts is demonstrated via the spike I did
- Business logic implemented using table modules is demonstrated through a separate code example included in the code examples.

3.4 Intended Learning Outcome 4

ILO 4: Explain security issues that arise when developing distributed and service based applications and how these security concerns can be addressed to mitigate potential threats.

I believe I have covered this intended learning outcome to an **exemplary** level. I initially wrote a short report about SQL injection (to meet this ILO to a “good” level), but stretched it to a full research report about a number of major security issues, and ways to avoid them using .NET technologies. I have put a large amount of effort into the research report and spent around two weeks writing it, researching the most common security issues and methods to resolve them. In addition to the research report, I have **also** written a short report about security concerns for web services.

In addition to both of these reports, the experience report written about my main piece has a detailed writeup on how I have protected the application from the most common security issues, referring back to the research report.

I believe all the work I’ve done should help make this ILO reach the exemplary level (or if not, at least the outstanding level!)

Experience Report

Experience Report

Table of Contents

1	Introduction	3
2	Problem Description	3
3	Requirements.....	3
4	Solution Architecture	4
5	Core Projects.....	4
5.1	Jobs.DataAccess	4
5.2	Jobs.BusinessLogic	5
5.3	Jobs.UI.Web	5
6	Unit Test Projects.....	5
6.1	Jobs.Tests.BusinessLogic.....	5
7	Additional Projects – Authentication.....	5
7.1	Jobs.AuthExample.Service	5
7.2	Jobs.AuthExample.Client	6
8	Security	6
8.1	User authentication	6
8.2	Most Common Security Issues.....	7
8.3	SQL Injection	7
8.4	Cross-Site Scripting	7
8.5	Cross-Site Request Forgery	7
8.6	Failure to restrict URL access	7
8.7	Username enumeration.....	7
9	Summary	8
10	References	8
11	Appendices.....	9
11.1	Class Diagrams	9
11.1.1	Data Access Layer.....	9
11.1.2	Business Logic Layer.....	10
11.1.3	UI Layer	11
11.1.4	Business Layer unit tests.....	12
11.2	Screenshots.....	12

11.2.1	Main listing and job details	12
11.2.2	Begin application.....	13
11.2.3	Create account and apply	13
11.2.4	Administration – Index.....	14
11.2.5	Administration – Viewing applications	14

1 Introduction

This experience report will cover the major piece I've developed as part of my portfolio. My employer (PageUp People) is a provider of a Software-as-a-Service system for careers websites. They host the careers website for a number of large Australian companies, including Coles, Optus, ANZ, BHP Billiton, Commonwealth Bank, Kmart, NAB and Suncorp. Their system has been around for over 10 years now, and so there is a large amount of legacy code in the system.

This portfolio piece is a demonstration of how the system *may* have been designed if it were to be redone using more modern methodologies. Of course, not all the functionality is present (as the system is MASSIVE), but some of the most common use cases have been completed:

- Recruiter listing a new job in the system
- Applicant applying for a job in the system
- Recruiter viewing all applications to a job

2 Problem Description

MegaCo¹, a large corporation, is often hiring for many roles at the same time. Currently, they accept job applications via email and fax. With the large number of applications they are receiving, this process is becoming harder and harder. It's impossible to keep track of all the applicants, and some good applications end up getting lost. MegaCo has hired a well-known recruitment provider, JobsRUs, to supply them with a computer-based RMS (Recruitment Management System), to help them manage all the jobs and applications.

3 Requirements

The jobs system has the following requirements:

- Recruiters can:
 - Log in as a recruiter
 - List new jobs
 - See applications to all the listed jobs
- Applicants can:
 - Register / Log in as an applicant
 - View all listed jobs

¹ Company name is fictitious ☺

- Apply for any listed job
- Update their profile
- Ability to integrate into third party systems, in several parts of the system. Examples:
 - Posting jobs to SEEK, MyCareer, etc.
 - Single sign-on for recruiters

4 Solution Architecture

The system is structured in a three-tiered design (Data Access Layer, Business Layer, UI Layer), using the ASP.NET MVC framework at the UI layer. Interfaces are used at all layers, and the Castle Windsor IoC container library is used for dependency injection. This allows you to easily swap out the business layer or data access layer implementation with another implementation, without having to change any code. As long as it implements the interfaces correctly, there would be no issues.

This architecture is relatively common for web applications. In the future, a web service could potentially be added for creating job applications and jobs, and it could move to a service-oriented architecture. I did not think this was necessary or in-scope for this simple solution, so I've stuck with a simple three-tiered design.

5 Core Projects

The solution is split into three main projects. Class diagrams for all these projects are available in the appendices.

5.1 Jobs.DataAccess

This is the project containing the data access layer, implemented using Entity Framework 4.1. Entity Framework was chosen because it is the latest data access technology from Microsoft, and it is quite easy to use (although it is quite complex, it is quite easy to get started with). It supports Code-First development, meaning there's no autogenerated code and you can write all the model code yourself instead. Code-First can generate the database schema based on the models, too.

Additionally, since it uses the ADO.NET database provider model, switching to a different database system is easy. I was using SQL Server Express (provided with Visual Studio) during development, and switched to SQL Server Compact 4.0 when deploying a test version to a test Windows Server 2008 R2 VPS I've got (as I don't have enough memory free to install SQL Server Express). Even though these are both dialects of SQL Server, they use different ADO.NET providers. Switching them was simply a matter of changing the connection string in the Web.config file. It may even be possible to switch to SQLite or MySQL, although I did not test this. I've written about the ADO.NET provider model in more detail in a separate short report.

The repository pattern is used for CRUD operations. This abstracts the CRUD code from the actual domain entities, which makes it easier to switch to a different database technology in the future. One advantage of using the repository pattern over the active record pattern is that the entities are "disconnected". They're simple objects rather than database objects, and hence they can be safely passed down to the Business logic and UI layers (without converting them to basic DTOs, and without fear of the business logic code or UI code calling any database methods on the models).

5.2 Jobs.BusinessLogic

This is the project containing the business logic layer. It is implemented using the domain model pattern. This is because the data access layer is using a domain model technology (Entity Framework), so the domain model pattern is a good fit. The domain model pattern is the pattern most commonly used with MVC websites, combined with a domain model data access layer technology.

5.3 Jobs.UI.Web

This is the project containing the user interface layer. It is implemented using the ASP.NET MVC framework. The ASP.NET MVC framework was chosen because it is easy to use, and generally results in cleaner code than ASP.NET web forms. Additionally, the code is a lot easier to unit test, so unit tests could be integrated later if wanted. Due to time constraints, I was unable to implement full unit tests in my project, I only wrote some tests for the business layer.

I've written more detail about the MVC pattern and the ASP.NET MVC framework elsewhere in my portfolio.

6 Unit Test Projects

6.1 Jobs.Tests.BusinessLogic

This is an incomplete unit test project for the Jobs.BusinessLogic solution. It is implemented using the Visual Studio Unit Testing Framework², which was chosen because it comes with Visual Studio (so no third party dependencies are required). The unit testing project also uses the Moq³ mocking library, which was chosen due to its simplicity and popularity in the .NET community.

It currently contains a few unit tests for the application and applicant business logic classes, testing their business rules are correctly applied. In the future, these could be extended to test more (as more business logic rules are added), and unit tests for other projects (such as the UI layer and authentication services) could be added.

7 Additional Projects – Authentication

In addition to these main projects, there are also two others that demonstrate implementing a custom ASP.NET membership provider:

7.1 Jobs.AuthExample.Service

This project is a simple WCF service that demonstrates how to make an authentication web service. During development, this service uses the standard wsHttp binding, as the server built-in to Visual Studio does not support HTTPS⁴. However, in a live environment (and on my live test server), this service would use the wsHttps binding, and hence be secured via HTTPS.

² <http://msdn.microsoft.com/en-us/library/ms243147.aspx>

³ <http://code.google.com/p/mog/>

⁴ See <http://stackoverflow.com/questions/4651431/rest-wcf-service-over-ssl/4652059#4652059>

Since it is just an example, it is not 100% complete. The basic functionality (checking for a valid username and password, and registering new users) is supported, but in a live environment, additional security measures (such as brute force protection) would be implemented.

7.2 Jobs.AuthExample.Client

This project is a basic example of how to implement a custom ASP.NET membership provider. It calls the service mentioned above (Jobs.AuthExample.Service) to actually do the authentication. Like the service, basic functionality (such as checking for a valid username and password, and registering new users) is supported. Only the bare minimum number of required methods are implemented in the membership provider, all others (currently unused by the application) throwing NotImplementedException.

A role provider is not implemented due to time constraints. In the future, roles could be used to differentiate between administrators and recruiters, for example.

8 Security

8.1 User authentication

User authentication is quite an important issue with this system. There are two unique types of users – The recruiters, and the applicants. Applicants will have a password, however they will just need it to check the progress of their application (and update it if needed) and will not really use it for anything else. Hence, the applicants do not need a full user authentication system, and so their password will just be stored (salted and hashed) in the applicant database table.

On the other hand, recruiters are different. They will be doing most of the work in the system – Creating new jobs, reviewing applicants for jobs, and so on. The company will most likely have an existing user authentication system in place (for example, Active Directory or another LDAP implementation, or something like Novell) which they will want to integrate into the jobs site. Because of this, I have chosen to use the standard .NET provider model, which is very easy to use in ASP.NET MVC. This allows us to easily switch the authentication model to use a different provider, without any changes to the code (as long as the provider exists, of course!).

As mentioned above, I've also written a sample authentication WCF service and consumer, as an example of how to implement a custom authentication provider. In the future, custom authentication providers could be used for things like Single Sign-On or some other sort of unified login system.

Actually authenticating the users is handled by the framework. With ASP.NET MVC, you simply need to add the [Authorize] attribute to any controllers (for the whole controller) or actions (for individual actions) to force the user to be authenticated in order to use them. If a user tries to access a controller and they're not logged in, they will be redirected to the login page. This also supports user roles, although I have not used roles in my application. In the future, it could be modified to have different roles (like system administrators, recruiters, hiring managers, etc.)

8.2 Most Common Security Issues

The OWASP top 10 security issues are a list of the top 10 most common security issues found in web applications (OWASP 2010a). I have attempted to ensure my code is not vulnerable to **any** of these security vulnerabilities. Listed below are the most common issues, and a description of the steps I've taken to ensure my application is not vulnerable

8.3 SQL Injection

SQL injection is discussed in detail in my security research report, so I will not go in to detail here. As I am using the Entity Framework, SQL injection is not an issue. As mentioned in the research report, one of the easiest methods of avoiding SQL injection vulnerabilities is to use a newer database technology. When using an ORM system like Entity Framework, you do not need to write any SQL yourself, which ensures that there will not be any security vulnerabilities in it.

8.4 Cross-Site Scripting

Cross-Site Scripting (XSS) is basically insertion of arbitrary script tags in the page, and is the result of displaying user-supplied data on the page without "encoding" it. The OWASP project mentions that "XSS is the most prevalent web application security flaw" (OWASP 2010b), because it is quite easy to accidentally introduce an XSS hole into your application.

By default, the new "Razor" templating engine in MVC 3 automatically encodes **all** outputted content (Guthrie 2010), ensuring Cross-Site Scripting is not possible. All text output in my application is done via the Razor templating engine.

8.5 Cross-Site Request Forgery

Cross-Site Request Forgery (XSRF) is somewhat similar to XSS (and XSS can be used as an attack vector). More information on XSRF can be found in the security research report.

My application is not vulnerable to XSRF because all forms have anti-forgery tokens. This is a standard feature of the ASP.NET MVC framework, as I've described in my security research report.

8.6 Failure to restrict URL access

Failure to restrict URL access basically means that users are able to access administration sections of the application, by guessing the URL. Sounds simple, but surprisingly it is the 8th most common security issue (OWASP 2010a). My application is not vulnerable to this issue because it uses the authentication functionality built-in to ASP.NET MVC (as mentioned in the user authentication section above). The authentication functionality ensures that controllers decorated with the "Authorize" attribute are only accessible if a user is logged in. If not, it kicks users back to a login page.

This is part of the framework and has been extensively tested, so there should not be any issues with it. Using a framework's built-in security features is almost always more secure than "rolling your own" security, and this is the case with ASP.NET (Anderson 2011).

8.7 Username enumeration

Username enumeration is a less common security issue. It is a type of attack where the backend validation script tells the attacker if the supplied username is correct or not. My application is not vulnerable because it displays a generic "The user name or password provided is incorrect" error

message if either the username or password is wrong, giving away no information as to whether the username is valid or not.

9 Summary

Using my knowledge of Enterprise .NET technologies, I have developed a solution using a three-tiered architecture combined with MVC for the UI layer. This solution is efficient, has low coupling, and is easy to modify and upgrade. Common security issues such as SQL injection, XSS (Cross Site Scripting), XSRF (Cross-Site Request Forgery), username enumeration and unrestricted administration access have been avoided through common sense, using the latest technologies, and knowledge of the most common issues that plague web applications.

There is a test version live at <http://109.169.68.230/test/>, although I can't guarantee this will be up all the time. The test administration username and password are "admin" and "password1" respectively. Note that the first load takes a while as the application gets stopped by IIS when it is idle (and I have not yet figured out how to stop that happening).

10 References

Anderson, R 2011, *Securing your ASP.NET MVC 3 Application*, viewed 2011-06-04, <<http://blogs.msdn.com/b/rickandy/archive/2011/05/02/securing-your-asp-net-mvc-3-application.aspx>>.

Guthrie, S 2010, *Introducing "Razor" - a new view engine for ASP.NET*, viewed 2011-05-21, <<http://weblogs.asp.net/scottgu/archive/2010/07/02/introducing-razor.aspx>>.

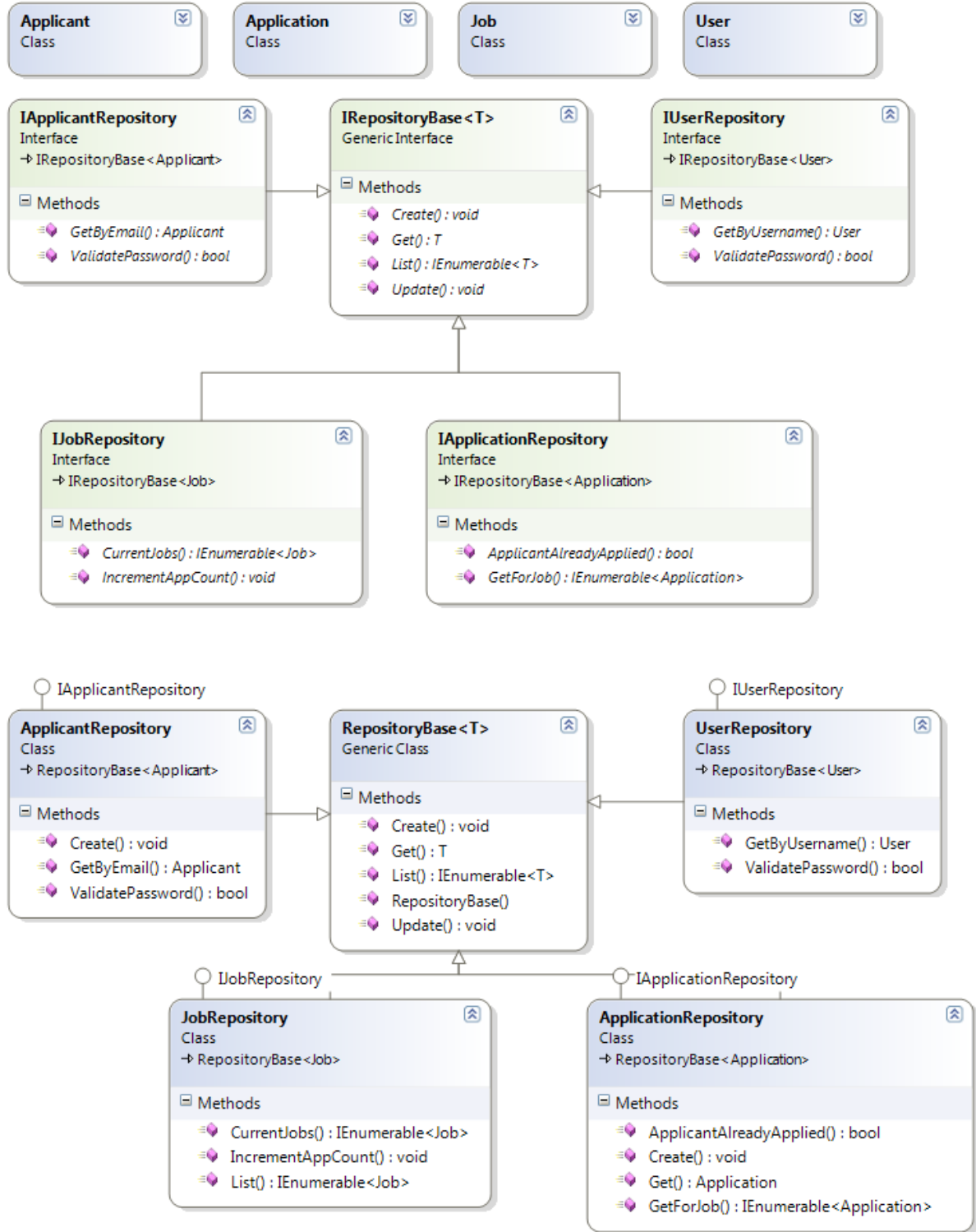
OWASP 2010a, *Open Web Application Security Project Top 10 Project*, viewed 2011-05-19, <https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project>.

OWASP 2010b, *Top 10 2010 - Cross-Site Scripting (XSS)*, viewed 2011-05-21, <https://www.owasp.org/index.php/Top_10_2010-A2>.

11 Appendices

11.1 Class Diagrams

11.1.1 Data Access Layer



11.1.2 Business Logic Layer

IApplicantsBL
Interface

Methods

- Create() : void
- Get() : Applicant
- GetByEmail() : Applicant
- Update() : void
- ValidatePassword() : bool

ApplicantsBL
Class

Fields

- _applicantRepository : IApplicantRepository

Methods

- ApplicantsBL()
- Create() : void
- Get() : Applicant
- GetByEmail() : Applicant
- Update() : void
- ValidatePassword() : bool

IApplicationBL
Interface

Methods

- Create() : void
- Get() : Application
- GetForJob() : IEnumerable<Application>

ApplicationBL
Class

Fields

- _applicationRepository : IApplicationRepository
- _jobRepository : IJobRepository

Methods

- ApplicationBL()
- Create() : void
- Get() : Application
- GetForJob() : IEnumerable<Application>

IUserBL
Interface

Methods

- Create() : bool
- GetByUsername() : User
- ValidatePassword() : bool

UserBL
Class

Fields

- _userRepository : IUserRepository

Methods

- Create() : bool
- GetByUsername() : User
- UserBL()
- ValidatePassword() : bool

IJobsBL
Interface

Methods

- Create() : void
- CurrentJobs() : IEnumerable<Job>
- Get() : Job
- List() : IEnumerable<Job>
- Update() : void

JobsBL
Class

Fields

- _jobRepository : IJobRepository

Methods

- Create() : void
- CurrentJobs() : IEnumerable<Job>
- Get() : Job
- JobsBL()
- List() : IEnumerable<Job>
- Update() : void

11.1.3 UI Layer

Controllers for normal users (Jobs.UI.Web.Controllers):

JobController
Class
→ Controller

Fields

- _jobs : IJobsBL

Methods

- Details() : ActionResult
- Index() : ActionResult
- JobController()

ApplyController
Class
→ Controller

Fields

- _applicantBL : IApplicantsBL
- _applicationBL : IApplicationBL
- _jobsBL : IJobsBL

Methods

- ApplyController()
- Begin() : ActionResult
- LogOn() : ActionResult
- NewApplicant() : ActionResult (+ 1 overload)
- Questions() : ActionResult (+ 1 overload)
- Success() : ActionResult
- UpdateApplicant() : ActionResult (+ 1 overload)

Controllers for recruiters/administrators (Jobs.UI.Web.Areas.Admin.Controllers):

ApplicationController
Class
→ Controller

Fields

- _applicationBL : IApplicationBL
- _jobsBL : IJobsBL

Methods

- ApplicationController()
- Index() : ActionResult
- View() : ActionResult

JobController
Class
→ Controller

Fields

- _jobs : IJobsBL

Methods

- Create() : ActionResult (+ 1 overload)
- Edit() : ActionResult (+ 1 overload)
- Index() : ActionResult
- JobController()

IWindsorInstaller

ControllersInstaller
Class

Methods

- FindAdminControllers() : BasedOnDescriptor
- FindControllers() : BasedOnDescriptor
- Install() : void

WindsorControllerFactory
Class
→ DefaultControllerFactory

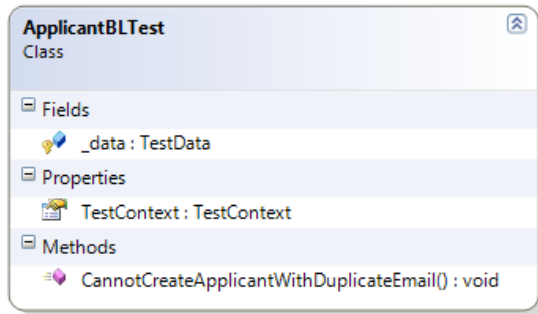
Fields

- _kernel : IKernel

Methods

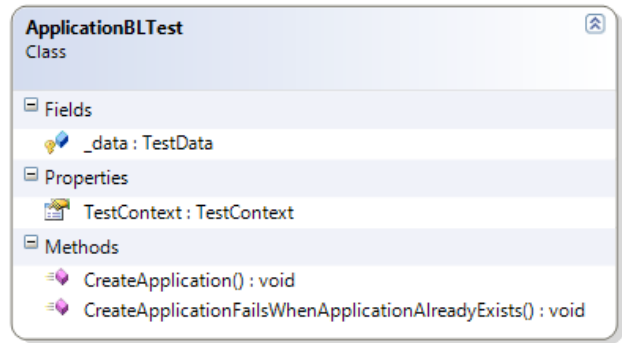
- GetControllerInstance() : IController
- ReleaseController() : void
- WindsorControllerFactory()

11.1.4 Business Layer unit tests



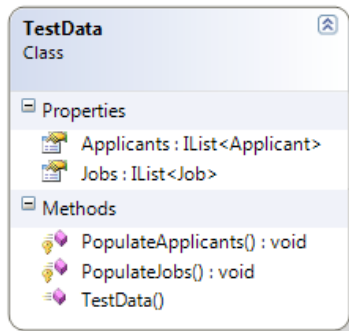
ApplicantBLTest
Class

- Fields
 - _data : TestData
- Properties
 - TestContext : TestContext
- Methods
 - CannotCreateApplicantWithDuplicateEmail() : void



ApplicationBLTest
Class

- Fields
 - _data : TestData
- Properties
 - TestContext : TestContext
- Methods
 - CreateApplication() : void
 - CreateApplicationFailsWhenApplicationAlreadyExists() : void

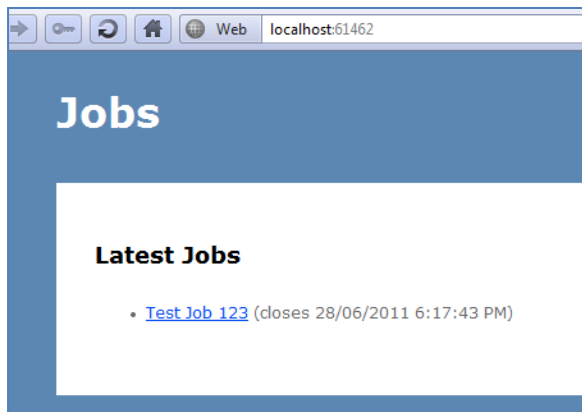


TestData
Class

- Properties
 - Applicants : IList<Applicant>
 - Jobs : IList<Job>
- Methods
 - PopulateApplicants() : void
 - PopulateJobs() : void
 - TestData()

11.2 Screenshots

11.2.1 Main listing and job details



Web localhost:61462

Jobs

Latest Jobs

- [Test Job 123](#) (closes 28/06/2011 6:17:43 PM)



Test Job 123

This is an awesome job!!!

Job closes 28/06/2011 6:17:43 PM

[Apply now!](#)

11.2.2 Begin application

Begin Your Application

In order to apply for a job, you need to have an account. [Create an account.](#)

Existing Applicant

Already have an account? Log in below:

Email

Password

11.2.3 Create account and apply

New Applicant

Please fill in all the below information to create an account.

First Name

Last Name

Email

Password

Address

Suburb

PostCode

State

Country

Application Form

First question would go here

Second question would go here

Any other comments to add to your application?

11.2.4 Administration – Index

Index

[Create New](#)

	Title	Opening Date	Closing Date	Application Count
Edit View Applications	Test Job 123	28/05/2011 6:17:43 PM	28/06/2011 6:17:43 PM	2

11.2.5 Administration – Viewing applications

Applications

	First Name	Last Name
View	Daniel	Lo Nigro
View	asd	asd

View Application of Daniel Lo Nigro

The Applicant

First Name: Daniel
Last Name: Lo Nigro
Email: daniel@dan.cx

Address:
274 Rossmoyne Street
Thornbury, Victoria 3071
Australia

The Application

First question would go here
This is the first answer

Second question would go here
Second answer

Any other comments to add to your application?
NO!!!


```
1 using System;
2 using System.Collections.Generic;
3 using Jobs.BusinessLogic.Abstract;
4 using Jobs.BusinessLogic.Exceptions;
5 using Jobs.DataAccess.Entities;
6 using Jobs.DataAccess.Repositories.Abstract;
7
8 namespace Jobs.BusinessLogic
9 {
10     public class ApplicationBl : ApplicationBl
11     {
12         protected IApplicationRepository _applicationRepository;
13         protected IJobRepository _jobRepository;
14
15         public ApplicationBl(IApplicationRepository applicationRepository, IJobRepository jobRepository)
16         {
17             _applicationRepository = applicationRepository;
18             _jobRepository = jobRepository;
19         }
20
21         public void Create(Application application)
22         {
23             // Business rule: Applicant can only have one application for a job
24             if (_applicationRepository.ApplicantAlreadyApplied(application.Applicant, application.Job))
25                 throw new DuplicateApplicationException("You have already applied for this job!");
26
27             _applicationRepository.Create(application);
28
29         }
30
31         // Business rule: When an application is received, increment application count for the job by 1
32         _jobRepository.IncrementAppCount(application.Job);
33     }
34
35     public IEnumerable<Application> GetForJob(Job job)
36     {
37         return _applicationRepository.GetForJob(job);
38     }
39
40     public Application Get(int applicationId)
41     {
42         return _applicationRepository.Get(applicationId);
43     }
44 }
```

C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099 ...Portfolio\Code\Jobs\Jobs.DataAccess\Entities\Job.cs 1

```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel.DataAnnotations;
4
5 namespace Jobs.DataAccess.Entities
6 {
7     public class Job
8     {
9         public int JobId { get; set; }
10
11         [Required]
12         [MinLength(5)]
13         public string Title { get; set; }
14
15         [Required]
16         public DateTime OpeningDate { get; set; }
17
18         [Required]
19         public DateTime ClosingDate { get; set; }
20
21         [DataType(DataType.MultilineText)]
22         [Required]
23         public string Description { get; set; }
24
25         public int ApplicationCount { get; set; }
26
27         public virtual ICollection<Application> Applications { get; set; }
28     }
29 }
30
```

```
1 using System.Collections.Generic;
2 using System.Linq;
3 using Jobs.DataAccess.Entities;
4 using Jobs.DataAccess.Repositories.Abstract;
5
6 namespace Jobs.DataAccess.Repositories
7 {
8     public class ApplicationRepository : RepositoryBase<Application>, ApplicationRepository
9     {
10         public override void Create(Application application)
11         {
12             using (var context = new JobsContext())
13             {
14                 // Tell the framework the job and applicant are not new
15                 context.Jobs.Attach(application.Job);
16                 context.Applicants.Attach(application.Applicant);
17
18                 context.Applicants.Add(application);
19                 context.SaveChanges();
20             }
21         }
22         public bool ApplicantAlreadyApplied(Applicant applicant, Job job)
23         {
24             using (var context = new JobsContext())
25             {
26                 int count = (from app in context.Applicants
27                             where app.JobId == job.JobId
28                             select app).Count();
29                 return count != 0;
30             }
31         }
32     }
33
34     public IEnumerable<Application> GetForJob(Job job)
35     {
36         using (var context = new JobsContext())
37         {
38             // Join to applicant as we want applicant details too.
39             var applications = from application in context.Applications.Include("Applicant")
40                               where application.JobId == job.JobId
41                               select application;
42
43             return applications.ToList();
44         }
45     }
46
47     public override Application Get(int id)
48     {
49         using (var context = new JobsContext())
50         {
51             return context.Applications.Include("Applicant").Where(a => a.ApplicationId == id).
52                 FirstOrDefault();
53         }
54     }
55 }
56
```

C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099 ... \Jobs\Jobs.DataAccess\Repositories\JobRepository.cs 1

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using Jobs.DataAccess.Entities;
5 using Jobs.DataAccess.Repositories.Abstract;
6
7 namespace Jobs.DataAccess.Repositories
8 {
9     public class JobRepository : RepositoryBase<Job>, IJobRepository
10    {
11        public IEnumerable<Job> CurrentJobs()
12        {
13            using (var context = new JobsContext())
14            {
15                var jobs = from job in context.Jobs
16                    where job.OpeningDate < DateTime.Now
17                    where job.ClosingDate > DateTime.Now
18                    orderby job.ClosingDate
19                    select job;
20
21                return jobs.ToList();
22            }
23        }
24
25        public void IncrementAppCount(Job job)
26        {
27            job.ApplicationCount++;
28            Update(job);
29        }
30
31        public override IEnumerable<Job> List()
32        {
33            using (var context = new JobsContext())
34            {
35                var items = from job in context.Jobs
36                    orderby job.JobId descending
37                    select job;
38
39                return items.ToList();
40            }
41        }
42    }
43 }
44
```

C:\Users\Daniel\Documents\My Dropbox\Uni\H113099 ... Jobs\Jobs.DataAccess\Repositories\RepositoryBase.cs 1

```
1 using System.Collections.Generic;
2 using System.Data;
3 using System.Linq;
4 namespace Jobs.DataAccess.Repositories
5 {
6     public class RepositoryBase<T> where T : class
7     {
8         public RepositoryBase()
9         {
10             //__context = new JobsContext();
11             //__set = __context.Set<T>();
12         }
13     }
14
15     public virtual void Create(T entity)
16     {
17         using (var context = new JobsContext())
18         {
19             context.Set<T>().Add(entity);
20             context.SaveChanges();
21         }
22     }
23
24     public virtual T Get(int id)
25     {
26         using (var context = new JobsContext())
27         {
28             return context.Set<T>().Find(id);
29         }
30     }
31
32     public virtual IEnumerable<T> List()
33     {
34         using (var context = new JobsContext())
35         {
36             var items = from T in context.Set<T>()
37                         select T;
38             return items.ToList();
39         }
40     }
41
42     public virtual void Update(T entity)
43     {
44         using (var context = new JobsContext())
45         {
46             context.Set<T>().Attach(entity);
47             context.Entry(entity).State = EntityState.Modified;
48             context.SaveChanges();
49         }
50     }
51
52     }
53 }
54
```

```

1 using System;
2 using System.Web.Mvc;
3 using Jobs.BusinessLogic.Abstract;
4 using Jobs.BusinessLogic.Exceptions;
5 using Jobs.DataAccess.Entities;
6 using Jobs.UI.Web.Models.Apply;
7
8 namespace Jobs.UI.Web.Controllers
9 {
10     public class ApplyController : Controller
11     {
12         protected IApplicantSBL _applicantSBL;
13         protected IJobsSBL _jobsSBL;
14         protected IApplicantSBL _applicantSBL;
15
16         public ApplyController(IApplicantSBL applicantSBL, IJobsSBL jobsSBL, IApplicantSBL applicantSBL)
17         {
18             _applicantSBL = applicantSBL;
19             _jobsSBL = jobsSBL;
20             _applicantSBL = applicantSBL;
21         }
22
23         /// <summary>
24         /// Begin an application
25         /// </summary>
26         /// <param name="id">Job ID</param>
27         /// <returns></returns>
28         public ActionResult Begin(int id)
29         {
30             ViewBag.JobId = id;
31             return View();
32         }
33
34         /// <summary>
35         /// Create a new applicant
36         /// </summary>
37         /// <param name="id">Job ID</param>
38         /// <returns></returns>
39         public ActionResult NewApplicant(int id)
40         {
41             return View();
42         }
43
44         /// <summary>
45         /// Save a newly created applicant
46         /// </summary>
47         /// <param name="id">Job ID</param>
48         /// <param name="applicant">Applicant details</param>
49         /// <returns></returns>
50         [HttpPost]
51         [ValidateAntiForgeryToken]
52         public ActionResult NewApplicant(int id, Applicant applicant)
53         {
54             // Return the form again if the model is invalid
55             if (!ModelState.IsValid)
56                 return View();
57
58             // Try to create the applicant
59             try
60             {
61                 _applicantSBL.Create(applicant);
62                 Session["applicantId"] = applicant.ApplicantId;
63                 return RedirectToAction("Questions", new { Id = id });
64             }
65             catch (BusinessLogicException ex)
66             {
67                 ModelState.AddModelError("BusinessLogic", ex.Message);
68                 return View();
69             }
70         }
71
72         /// <summary>
73         /// Show the application form questions
74         /// </summary>

```

```

75         /// <param name="id">ID of the job</param>
76         /// <returns></returns>
77         public ActionResult Questions(int id)
78         {
79             // If there's no applicant ID in the session, redirect to the start
80             if (Session["applicantId"] == null)
81                 return RedirectToAction("Begin", new { Id = id });
82
83             return View();
84         }
85
86         /// <summary>
87         /// Save the application form answers
88         /// </summary>
89         /// <param name="id">ID of the job</param>
90         /// <param name="application">Application data</param>
91         /// <returns></returns>
92         [ValidateAntiForgeryToken]
93         [HttpPost]
94         public ActionResult Questions(int id, Application application)
95         {
96             // If there's no applicant ID in the session, redirect to the start
97             if (Session["applicantId"] == null)
98                 return RedirectToAction("Begin", new { Id = id });
99
100             // Model not valid? Redisplay form
101             if (!ModelState.IsValid)
102                 return View();
103
104             // Set the applicant and the job
105             application.Job = _jobsSBL.Get(id);
106             application.Applicant = _applicantSBL.Get(Convert.ToInt32(Session["applicantId"]));
107
108             // Try to create the application
109             try
110             {
111                 _applicantSBL.Create(application);
112                 return RedirectToAction("Success");
113             }
114             catch (BusinessLogicException ex)
115             {
116                 ModelState.AddModelError("BusinessLogic", ex.Message);
117                 return View();
118             }
119
120             public ActionResult Success()
121             {
122                 return View();
123             }
124
125             [HttpPost]
126             [ValidateAntiForgeryToken]
127             public ActionResult Login(int id, LoginModel model)
128             {
129                 ViewBag.JobId = id;
130                 if (!ModelState.IsValid)
131                     return View();
132
133                 // Try find applicant
134                 Applicant applicant = _applicantSBL.GetByEmail(model.Email);
135                 if (applicant == null)
136                 {
137                     ModelState.AddModelError("Email", "Your email address or password was incorrect");
138                     return View();
139                 }
140                 if (!applicantBL.ValidatePassword(applicant, model.Password))
141                 {
142                     ModelState.AddModelError("Password", "Your email address or password was incorrect");
143                     return View();
144                 }
145             }
146
147
148

```

```
149     Session["applicantId"] = applicant.ApplicantId;
150     return RedirectToAction("UpdateApplicant", new { Id = id });
151 }
152
153     public ActionResult UpdateApplicant(int id)
154     {
155         Applicant applicant = applicantBL.Get(Convert.ToInt32(Session["applicantId"]));
156         return View(applicant);
157     }
158
159     [HttpPost]
160     public ActionResult UpdateApplicant(int id, [Bind(Exclude="ApplicantId")] Applicant
161         applicantChanges)
162     {
163         // If there's no applicant ID in the session, redirect to the start
164         if (Session["applicantId"] == null)
165             return RedirectToAction("Begin", new { Id = id });
166
167         // We need to load the old applicant (so it's tracked by Entity Framework) and copy
168         // over all the modified details
169         Applicant oldApplicant = _applicantBL.Get(Convert.ToInt32(Session["applicantId"]));
170         applicantChanges.Password = oldApplicant.Password;
171
172         // Return the form again if the model is invalid
173         if (!ModelState.IsValid)
174             return View(applicantChanges);
175
176         UpdateModel(oldApplicant);
177
178         // Try to update the applicant
179         try
180         {
181             _applicantBL.Update(oldApplicant);
182             return RedirectToAction("Questions", new { Id = id });
183         }
184         catch (BusinessLogicException ex)
185         {
186             ModelState.AddModelError("BusinessLogic", ex.Message);
187             return View(applicantChanges);
188         }
189     }
190 }
191 }
192 }
```


Security

Research Report

Research Report – Security

1 Abstract

This report will explain the various security issues that are common in today's web applications (such as SQL injection, XSS and XSRF attacks), give some examples of the vulnerabilities, and how they could be prevented. I've had experience developing web applications for a relatively long time, and have read a large amount of literature pertaining to application security to prepare this report.

2 Introduction

Security is a very important issue for ALL enterprise applications. Users are trusting you with their data, so you should put some effort into protecting the integrity of said data (and, depending on the situation, you may be legally obliged to). As demonstrated by Sony recently¹, even software systems by large companies can be affected by security issues. Unfortunately, it is far too common to see major security holes in applications, caused by small issues, or bad code that is overlooked during development and code reviews. A lot of these security issues are issues which could be resolved very easily by knowledge of the major issues affecting applications.

In this research report, I will briefly talk about some important security aspects of modern applications, as well as some of the most common security issues, and how they can be avoided.

¹ See <http://www.theage.com.au/world/gamers-details-stolen-in-sony-security-breach-20110427-1dwyr.html> and <http://arstechnica.com/tech-policy/news/2011/06/sony-hacked-yet-again-plaintext-passwords-posted.ars>

Table of Contents

1	Abstract.....	2
2	Introduction	2
3	Passwords	4
3.1	Hashing.....	4
3.2	Salt	5
3.2.1	System-wide salt	5
3.2.2	Per-user salt	6
4	Configuration file encryption	7
5	SQL Injection	8
5.1	Blind SQL injection	8
5.2	Direct SQL injection.....	9
5.3	Prevention.....	10
5.3.1	Prepared statements	10
5.3.2	Stored procedures.....	10
5.3.3	Newer technologies	10
6	Cross-Site Scripting	10
6.1	Prevention.....	11
6.1.1	HTML encoding	11
6.1.2	HTML Sanitisation	12
7	Cross-Site Request Forgery	12
7.1	Prevention.....	12
8	Conclusion.....	13
9	References	14

3 Passwords

Most modern systems store user credentials in some form, in order to authenticate that the user accessing the system is a valid user (and that they are who they say they are). Although there are many newer technologies (like OpenID and single sign-on systems), today the most common form of user credentials are usernames and passwords stored directly in your database. This is because they are easy to store, and there is no reliance on any third-party systems.

Storing these credentials correctly is an important step to ensuring your system has good security. The main method of ensuring password security is hashing the passwords.

3.1 Hashing

To discuss hashing, we must first discuss encryption. Encryption usually involves using a “key” to protect data, rendering it readable only if you have that key available, and you know the algorithm that was used to encrypt the data. However keys are not always used. A very simple example of encryption is the ROT13 algorithm. ROT13 simply “rotates” every alphabetic character 13 characters (Wikipedia 2011b). ROT13 is not secure at all, but does demonstrate what encryption is – “undoing” or reversing encryption is possible, you just need to know the algorithm that was used (and the key that was used, if the algorithm uses a key).

Hashing is a lot like encryption, except it is one-way. A hash is a unique string of characters that **cannot** be easily reversed back into a password. A simple hashing algorithm might be to get the ASCII value of every character, and add them together (to get a total). The total would be the hash of the string. Obviously, you can try guessing the original password, but you **cannot** retrieve the original password from this “hash” value. Again, this is a very insecure example, but is sufficient to describe what a hash is. In real scenarios, hashing algorithms are considerably more secure.

With secure hashing algorithms, each string has a unique hash, and even a very small change in the password produces a significant change in the hash output (see Figure 1 below). There are several well-known hashing algorithms, including MD5 and SHA1. For best security, it’s suggested to use a well-known hashing algorithm, as these have been tested for security. A common security issue is to make your own hashing algorithm and assume it’s more secure, as it’s custom. Unless you’ve studied cryptography in great detail, it’s almost certain that a proper hashing algorithm is significantly more secure than your own.

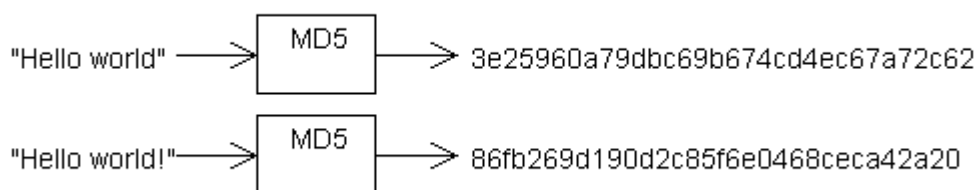


Figure 1: Example of the input and output to a hashing algorithm. Notice the addition of an exclamation mark to the string changes the hash significantly

The benefit of using a hash to protect user passwords is that if an attacker ever gets access to your database, they **never** see your user’s passwords, just the hash for their passwords. This ensures the passwords are safe. When validating user credentials, you simply re-hash their password, and check if the hashes match. If they don’t, the password is incorrect

3.2 Salt

There is one main vulnerability with hashing: Hashing the same string will always return the same hash. If you search Google for “3e25960a79dbc69b674cd4ec67a72c62” (the MD5 hash of “Hello world”), you’ll find several results referring to it. This is by design – You need to be able to rehash the password in order to check whether the user-entered password was correct. This can be exploited to make a very large list of the hashes of all the most common passwords, and using this precomputed table to look up the hashes of common passwords from a database. This is known as a “rainbow table”, and is a common threat to password hashing (Oechslin 2003).

To be immune from rainbow table attacks, an approach known as “salting” can be used. “Salting” is basically adding a unique string to the password before hashing it. So, instead of hashing the password, you hash the password combined with a salt string in some way. The salt string can be added to the beginning, the end, or somewhere in the middle of the password. As long as you’re consistent, it doesn’t matter where the salt is added. Since the input string before hashing is now significantly longer and won’t be a dictionary word (see Figure 2 below), this reduces the usefulness of rainbow tables.

There are two main ways to use salting:

3.2.1 System-wide salt

The first is to have a single system-wide salt. This is often specified in a configuration file in the system. Even this basic level of salting increases security quite a lot. Without a salt, an attacker only needs access to your database to try and reverse your user’s passwords (either via rainbow tables, or brute force). When a system-wide salt is used, the attacker **also** needs access to your file system (as the salt is stored in a configuration file).

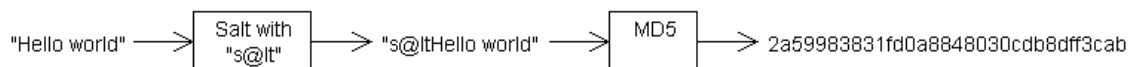


Figure 2: Example of input/output for hashing algorithm when a single salt is used

A simple example of a system-wide hash in C# would be something like this:

```

1  public class PasswordHasher
2  {
3      protected const string PASSWORD_SALT = "s@!t!sg0oDf0rY0u";
4
5      public static string HashPassword(string password)
6      {
7          // This is the important bit - Add the salt to the password!
8          string saltedPassword = password + PASSWORD_SALT;
9
10         HashAlgorithm hasher = new SHA256Managed();
  
```

```

11     byte[] hashBytes = hasher.ComputeHash(Encoding.ASCII.GetBytes(saltedPassword));
12     return Convert.ToBase64String(hashBytes);
13 }
14 }

```

This has already added quite a bit of security to the password hashing. However, if an attacker **does** gain access to both the database and the file system, they can find your salting algorithm and create a rainbow table that uses it. Additionally, if two users are using the same password, their hashes will still be identical. Using a per-user hash solves these two issues.

3.2.2 Per-user salt

An even better approach to salting is to have a per-user salt **in addition to** the system-wide salt (see Figure 3 below). For example, put the system-wide salt at the end of the password, and the per-user salt at the beginning of the password. The result of this is that each user's password hash will be different, even if their passwords are identical. Since the hashing algorithm is now unique for every single user, rainbow tables are rendered useless. At best, a rainbow table would only be able to decode the password of a single user (although this is very unlikely due to the salting).

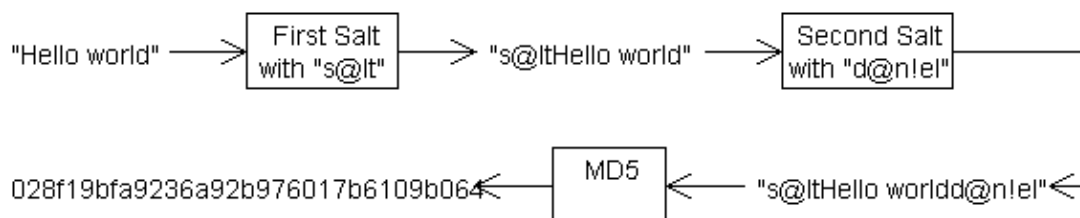


Figure 3: Example of input/output for a hashing algorithm when two salts are used

An example of this in C# might be something like the following:

```

1  public class PasswordHasher
2  {
3      protected const string PASSWORD_SALT = "s@!t!sg0oDf0rY0u";
4
5      public static string HashPassword(string username, string password)
6      {
7          HashAlgorithm hasher = new SHA256Managed();
8
9          // Hash username to use as a salt
10         byte[] usernameBytes = hasher.ComputeHash(Encoding.ASCII.GetBytes(username));
11         string hashedUsername = Convert.ToBase64String(usernameBytes);
12
13         // Add hashed username as well as "system-wide" salt
14         string saltedPassword = password + hashedUsername + PASSWORD_SALT;
15
16         byte[] hashBytes = hasher.ComputeHash(Encoding.ASCII.GetBytes(saltedPassword));
17         return Convert.ToBase64String(hashBytes);
18     }
19 }

```

First the username is hashed normally (lines 10-11), and the hash of the username is then used to salt the password (line 14).

Of course, this is just a very simple example. In a live environment, you would most likely generate a random salt value for each user, and store this in the database in the user’s row. If you’re extra paranoid about security, you may use column encryption in SQL Server to encrypt the salt column. This is not necessary most of the time, however. Just using a salt is already significantly more secure than normal hashing, which is itself significantly more secure than storing passwords in plaintext.

4 Configuration file encryption

Often, configuration files contain sensitive data such as connection strings for databases, and salts for passwords (as mentioned in section 3.2 above). One major security issue is the exposure of sensitive information. This recently happened to microblogging website Tumblr – Their main configuration file leaked due to a mistyped PHP tag². If a configuration file is leaked, sensitive information such as database and API passwords can fall into the hands of hackers.

One method to protect this sensitive data is to use configuration file encryption. Configuration file encryption allows you to encrypt certain sections of the configuration file (Web.config). It is built in to ASP.NET 2.0 and higher (Prosis 2005), which means it can be used on any ASP.NET hosting provider and no additional components are required to use it.

Encryption is done using the **machine key**, which is unique per computer. Configuration files encrypted on one computer can only be decrypted on that same computer. This means that even if your configuration file is stolen, the attacker will not be able to decrypt it (Burnett & Foster 2004).

Configuration file encryption is very simple to do. For this example, I’ll use the following appSettings section:

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
  </system.web>
  <appSettings>
    <add key="Message" value="Hello World!" />
  </appSettings>
</configuration>
```

The “Message” value can be used from an ASP.NET WebForms page like this:

```
_message.Text = ConfigurationSettings.AppSettings["Message"];
```

To encrypt the appSettings section of the above configuration file, you simply need to run the following command in the directory of the website:

```
aspnet_regiis -pef appSettings . -prov DataProtectionConfigurationProvider
```

This will encrypt the file and should display a “Succeeded!” message. Note: If you don’t have command-line access (for example, if it is on a shared web host), you can instead create your own application which uses the encryption APIs to encrypt the config file.

On my machine, the above configuration file now looks like this:

² http://www.reddit.com/r/PHP/comments/g70iy/mistyped_tag_leads_to_exposure_of_tumblr_db/

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
  </system.web>
  <appSettings configProtectionProvider="DataProtectionConfigurationProvider">
    <EncryptedData>
      <CipherData>
        <CipherValue>AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAY2R19c3220iZSpoj4yBNrgQAAACAAAAAAQZgAAAAEAACAAADrPnysT
        qPo/Qtjq10v2E10QnTk6/iX6LxcD67mJH0hcQAAAAA0gAAAAIAACAAAAAc/2zcILuMiqq/oMp58rCSE4dCindBXPW9fcp+48Z8VKA
        AAAC1PRNzGHxbG2mfUAFjIoYud4T0NxD19dLtxSeaVaYQsU+twAH2NpInDmipYi1IWLLxjD7jI0T1ci3LgD28FNqpOX0dqs1+1wgp
        qLZadbOBizrq/aLQrETKmwKjVXvKVi918+BhFBM8EsC6ZFkPmba3E4I/RaBz7ir2/ijjLY0yI+e02WCHVcG9VD3amz5iCbvEythKfJ
        w/L1cMbnQucQBQAAAKJ1dXaQ1vn1+0I52thLqz4mF6HzosvJMKFPvN2JDhcM1jN+wxswSph8W9CIDmPD+mJv1sJ+EpzL5bqYoZZBH
        MA=</CipherValue>
      </CipherData>
    </EncryptedData>
  </appSettings>
</configuration>
```

As you can see, the raw value is no longer visible – It has been encrypted. ASP.NET knows how to handle encrypted configuration files, and will automatically decrypt the configuration data when you access the website. In other words, **no** code changes are required at all!

5 SQL Injection

SQL injection used to be one of the most common security issues, and unfortunately, still is quite a big issue. In fact, it is consistently the #1 most common issue in OWASP (Open Web Application Security Project)'s list of top 10 web application security risks (OWASP 2010a). SQL injection is the result of using a specific type of vulnerability that involves using one language being dynamically constructed by another (in this case, it's using SQL inside a programming language, most likely C# in our case). Incorrect escaping of user-supplied data can lead to vulnerabilities in the formed SQL string, and arbitrary user-supplied SQL being executed.

There are several different types of SQL injection, including:

5.1 Blind SQL injection

Blind SQL injection is when you can inject arbitrary SQL, but can't use it to view any rows from the database (Halfond, Viegas & Orso 2006). The blind SQL injection example (located in the portfolio appendices, and in the "SqlInjection" folder on the provided CD) demonstrates one type of blind SQL injection. The vulnerable code is using the following method call to generate the SQL for validating a user's username and password:

```
string.Format(@"
  SELECT COUNT(*)
  FROM Users
  WHERE username = '{0}' AND password = '{1}'", username, password),
```

The username and password are being added to the query unmodified (let's ignore the issue that the password is not encrypted for now!). On the surface, this appears to work correctly, with invalid users being rejected, and valid users being allowed in:


```
Username: blah
Password: foo
Invalid username or password!
Username: admin
Password: p@ssw0rd
Welcome, admin!
```

However, the code is easily exploited with a specially-crafted username and password:

```
Username: admin
Password: ' OR '1'='1
Welcome, admin!
```

The user has just logged in without knowing the password for the “admin” account! But what happened, exactly? When the query was generated, it became the following:

```
SELECT COUNT(*) FROM Users WHERE username = 'admin' AND password = '' OR '1'='1'
```

Because 1 will always equal 1 (and OR has a higher precedence than AND), the password check is being totally bypassed!

5.2 Direct SQL injection

Direct or “normal” SQL injection is similar to blind SQL injection, except you can actually see the result of the injected query (i.e. rows are returned). This is often the type of SQL injection vulnerability found on listing pages (grid pages). These are often more severe than blind SQL injection vulnerabilities, as they can reveal sensitive data from the database (via injection of UNION queries).

A simple example is shown in the “DirectSqlInjection” code (in the appendices, and on the provided CD). In the example, the vulnerability is almost identical to the vulnerability in the “Blind SQL injection” section above – User-supplied data is being used without sanitisation. The program does work correctly:

```
Select type: phone
Mozart 7 by HTC (ID = 1)
Desire by HTC (ID = 2)
iPhone 3Gs by Apple (ID = 3)
iPhone 4 by Apple (ID = 4)
```

But it also allows SQL injection!

```
Select type: blah' UNION SELECT '1','1','test
test by 1 (ID = 1)
```

This is a very simple example of “union injection”. Once an SQL injection hole of this variety is found, it can be used to retrieve **any** data from the database that the database user has access to (Anley 2002). A lot of insecure sites run as the “sa” user, which gives you **full administrative access** to the database. Using UNION queries can be used to retrieve information about the database schema, including listing all the tables in the database. This is done via the inbuilt “information_schema” or “sys” views:

```
Select type: blah' UNION SELECT '1','1',table_name FROM information_schema.table
s --
Gadget by 1 (ID = 1)
```

GadgetType by 1 (ID = 1)
 Manufacturer by 1 (ID = 1)

It's even possible to get a list of all the other databases on the server, via the sys.databases view!

5.3 Prevention

Fortunately, prevention of all SQL injection attacks is very simple. SQL injection is really only an issue when dynamically constructing SQL yourself. Hence, there are several solutions:

5.3.1 Prepared statements

Prepared statements are SQL statements that have “placeholders” for all user-supplied arguments. As the arguments are specified separately from the SQL statements themselves, the database system knows how to correctly handle them (as it knows that they're all user-supplied inputs), and does all the required escaping. Using ADO.NET, this can be achieved using the **DbCommand** class³, as shown in the fixed blind SQL injection example.

5.3.2 Stored procedures

Stored procedures are like functions in a normal programming language. Like ordinary functions or methods, they accept parameters for user-supplied data. Much like prepared statements, the database system knows how to handle these and escape them correctly when used in SQL statements.

5.3.3 Newer technologies

Using basically **any** newer database technology is also a mitigation technique for SQL injection attacks. Technologies that abstract writing of the SQL away from the user (e.g. Entity Framework, LINQ to SQL, NHibernate, etc.) are not vulnerable to SQL injection attacks, as they properly handle user input. The backend code for these technologies will commonly use prepared statements, ensuring parameters are escaped correctly.

6 Cross-Site Scripting

Cross-Site Scripting (XSS) involves insertion of arbitrary HTML tags (such as `<script>` or `<iframe>` tags) in the page, and is the result of displaying user-supplied data on the page without “encoding” it.

Generally, XSS holes result are caused by using user input (for example, query string or form parameters) without sanitising them. When an XSS hole is present, a malicious user is able to inject their own HTML into the page.

The OWASP project mentions that “XSS is the most prevalent web application security flaw” (2010b), because it is quite easy to accidentally introduce an XSS hole into your application. XSS holes have

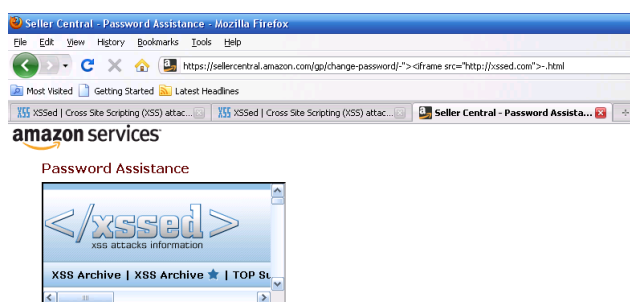


Figure 4: Example of an XSS hole in Amazon.com, with an `<iframe>` being injected. Credit: XSSed.com

³ <http://msdn.microsoft.com/en-us/library/system.data.common.dbcommand.aspx>

previously been found on large sites such as MySpace, eBay, Amazon, McAfee, American Express, and many more (XSSed.com 2011).

A simple example is the following ASP.NET code:

```
<p>Hello <%= Request.QueryString["name"] %></p>
```

This code outputs the “name” querystring parameter to the page. It is designed to be used by accessing the page with a URL similar to **Filename.aspx?name=Daniel**, which will work as expected. However, the page is not doing any validation or sanitisation of the input. If you go to **Filename.aspx?name=<script>alert('Hello World!')</script>**, the script tag will be written to the page, causing an alert with the text “Hello World” to appear:

```
<p>Hello <script>alert('Hello World!')</script></p>
```

This is just a harmless example, but more severe hacks can definitely be done. One of the most common attacks done via XSS is cookie stealing. A script can be injected that reads **document.cookie** (this is the cookie in JavaScript) and transmits it to the attacker’s server (Zuchlinski 2003). Depending on the server-side session security, the attacker might be able to log in as you using only your cookie!

6.1 Prevention

There are several mitigation techniques for cross-site scripting. The simplest solution is to simply **HTML encode** any user-supplied data that is displayed on the page.

6.1.1 HTML encoding

HTML encoding is basically the transformation of special HTML characters (like angle brackets < >) into “safe” characters, by using HTML encoding. For example, < becomes < and > becomes >. The encoded versions don’t have any special meaning in HTML, and hence they just display as normal characters.

The original way to safely display user-supplied data in ASP.NET was to call `Server.HtmlEncode` manually:

```
<p>Hello <%= Server.HtmlEncode(Request.QueryString["name"]) %></p>
```

This works, but it’s a bit verbose, and developers can easily forget to include the `HtmlEncode` call (leaving the web application open to Cross-Site Scripting) . In ASP.NET 4.0, a new syntax was added to make this easier (Guthrie 2010b). Instead of calling `Server.HtmlEncode` manually, you can simply use this syntax:

```
<p>Hello <%= Request.QueryString["name"] %></p>
```

And the text will automatically be HTML encoded.

The above examples all use the ASP.NET web forms “view engine” (used by ASP.NET web forms, and previous versions of ASP.NET MVC). ASP.NET MVC 3 comes with a new view engine called “Razor”. This new view engine automatically encodes **all** outputted content by default (Guthrie 2010a), similar to the “<%=” tag in ASP.NET web forms 4.0. The equivalent to the above using Razor would be:

```
<p>Hello @Request.QueryString["name"]</p>
```

6.1.2 HTML Sanitisation

HTML escaping is not appropriate in all situations. For example, if you are writing a blog system, you might want to allow people to use a subset of HTML in comments. In these cases, instead of encoding all HTML, you'd instead **cleanse** or **sanitise** the HTML – Keep any wanted HTML, and remove the rest. Generally, this is done by having a whitelist. Any HTML tags not in the whitelist are stripped from the text (Ter Louw & Venkatakrisnan 2009). One library that does this is the Microsoft XSS library, but there are many others. Another approach would be to implement sanitisation yourself, using a HTML parsing library such as the HTML Agility Pack⁴

7 Cross-Site Request Forgery

Cross-Site Request Forgery (XSRF) is somewhat similar to XSS (and XSS can be used as an attack vector). Essentially, XSRF is when another site transmits unauthorised commands to your website. The attack works by including a link, script or image in a page that “accesses a site to which the user is known (or is supposed) to have been authenticated” (Wikipedia 2011a).

While the HTTP standards state that GET requests are “safe”, idempotent and should not cause any side effects, they are quite often still used for requests that modify data. If your site uses GET requests in this manner, and you do not have any security tokens in the URL, you are vulnerable to XSRF attacks. A simple example would be a URI like **Delete.aspx?ID=123**. If Delete.aspx does not show any “Are you sure you want to delete this item?” verification message, an attacker could simply create a specially-crafted image tag that hits this URL:

```

```

Whenever someone that is authenticated on your site hits the attackers site, the tag will request that URL from your server. Since the user is logged in to your site, the request will succeed and item ID 123 will be deleted!

However, just using POST requests does not automatically protect you from XSRF attacks. Some attack vectors (e.g. using tags as mentioned above) are prevented; however, a malicious script can simply create an invisible form and post it to your page.

7.1 Prevention

Prevention of XSRF attacks is quite easy. One of the most common methods of XSRF prevention is to have a hidden, random, user-specific “token” (Shiflett 2004). Store the token in the session (or encrypted in a cookie), and also include it as one of the form fields (as a hidden value). When you validate the form submission, also validate that the token is correct (the one in the form post matches the one you have stored). Since the attacking site can't possibly know the correct token value, it can't send a valid request to your page.

Fortunately, in most modern web development frameworks you do not need to code this manually, and the ASP.NET MVC framework is no exception. Using XSRF protection in ASP.NET MVC is very simple. First, you need to include the “token” value in your form:

⁴ <http://htmlagilitypack.codeplex.com/>

```
@Html.AntiForgeryToken() // For a Razor view  
<%= Html.AntiForgeryToken() ?> // For standard ASP.NET views
```

Second, you need to decorate your action method with the “ValidateAntiForgeryToken” attribute:

```
[ValidateAntiForgeryToken]  
[HttpPost]  
public ActionResult DoSomething(StuffViewModel stuff)
```

This is all you need to do! Once this is done, ASP.NET will handle creating and validating the anti-forgery token. If the token is not found in the form post, an exception will be thrown.

8 Conclusion

Some of the most common security issues (such as SQL injection, XSS and XSRF attacks) account for the large majority of security issues in today’s web applications. These can all be overcome relatively easily via common sense, knowledge of the issues, and proper testing of your application. While not directly security issues as such, password hashing and configuration file encryption are important to ensure the confidentiality of your data (especially in case a security issue is found!).

There are many more security issues in today’s web applications – This report has only touched on a few of the most common. With proper penetration testing, web applications can be assured of a certain level of security, and you won’t end up like Sony! :-).

9 References

- Anley, C 2002, *Advanced SQL injection in SQL Server applications*, viewed 2011-05-19, <<http://sparrow.ece.cmu.edu/group/731-s11/readings/anley-sql-inj.pdf>>.
- Burnett, M & Foster, JC 2004, *Hacking the code: ASP. NET web application security*, Syngress Media Inc, viewed 2011-05-21,
- Guthrie, S 2010a, *Introducing "Razor" - a new view engine for ASP.NET*, viewed 2011-05-21, <<http://weblogs.asp.net/scottgu/archive/2010/07/02/introducing-razor.aspx>>.
- Guthrie, S 2010b, *New <%: %> Syntax for HTML Encoding Output in ASP.NET 4 (and ASP.NET MVC 2)*, viewed 2011-05-26, <<http://weblogs.asp.net/scottgu/archive/2010/04/06/new-lt-gt-syntax-for-html-encoding-output-in-asp-net-4-and-asp-net-mvc-2.aspx>>.
- Halfond, W, Viegas, J & Orso, A 2006, *A classification of SQL-injection attacks and countermeasures*, Citeseer, viewed 2011-06-04, <<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.2968&rep=rep1&type=pdf>>.
- Oechslin, P 2003, 'Making a faster cryptanalytic time-memory trade-off', *Advances in Cryptology-CRYPTO 2003*, pp. 617-630,
- OWASP 2010a, *Open Web Application Security Project Top 10 Project*, viewed 2011-05-19, <https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project>.
- OWASP 2010b, *Top 10 2010 - Cross-Site Scripting (XSS)*, viewed 2011-05-21, <https://www.owasp.org/index.php/Top_10_2010-A2>.
- Prosize, J 2005, *Five Undiscovered Features on ASP.NET 2.0*, viewed 2011-05-30, <<http://msdn.microsoft.com/en-us/magazine/cc163849.aspx>>.
- Shiflett, C 2004, *Cross-Site Request Forgeries*, viewed 2011-05-28, <<http://shiflett.org/articles/cross-site-request-forgeries>>.
- Ter Louw, M & Venkatakrisnan, VN 2009, 'Blueprint: Robust Prevention of Cross-site Scripting Attacks for Existing Browsers,' IEEE Computer Society,
- Wikipedia 2011a, *Cross-site request forgery - Wikipedia, The Free Encyclopedia*, viewed 2011-05-28, <http://en.wikipedia.org/wiki/Cross-site_request_forgery>.
- Wikipedia 2011b, *ROT13 - Wikipedia, The Free Encyclopedia*, viewed 2011-05-27, <<http://en.wikipedia.org/wiki/ROT13>>.
- XSSed.com 2011, *XSSed | Cross Site Scripting (XSS) attacks information and archive*, viewed 2011-06-05, <<http://xssed.com/>>.
- Zuchlinski, G 2003, *The Anatomy of Cross Site Scripting*, viewed 2011-06-01, <http://www.net-security.org/dl/articles/xss_anatomy.pdf>.

```

1 using System;
2 using System.Data.SqlClient;
3
4 namespace SQLInjection
5 {
6     /// <summary>
7     /// Program to demonstrate blind SQL injection vulnerabilities and how they can be avoided
8     /// </summary>
9     class Program
10    {
11        protected const string CONNECTION_STRING = @"
12            Data source=DANIEL-LAPTOP2\SQLEXPRESS;Initial Catalog=Example;
13            Integrated Security=True;Pooling=False";
14
15        protected static SqlConnection _connection;
16
17        static void Main(string[] args)
18        {
19            ConnectToDatabase();
20
21            string username;
22            do
23            {
24                Console.WriteLine("Username: ");
25                username = Console.ReadLine();
26                Console.WriteLine("Password: ");
27                string password = Console.ReadLine();
28                if (VulnerableCheck(username, password))
29                    //If (FixedCheck(username, password))
30                    {
31                        break;
32                    }
33
34                Console.WriteLine("Invalid username or password!");
35                Console.WriteLine();
36            } while (true);
37
38            Console.WriteLine("Welcome, {0}!", username);
39            Console.ReadKey();
40        }
41
42        protected static void ConnectToDatabase()
43        {
44            _connection = new SqlConnection(CONNECTION_STRING);
45            _connection.Open();
46        }
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
    /// <summary>
    /// Check username and password are valid. NOT vulnerable to SQL injection!
    /// </summary>

```

```

75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
    /// <param name="username">The username.</param>
    /// <param name="password">The password.</param>
    /// <returns>True if user is valid, otherwise False</returns>
    protected static bool FixedCheck(string username, string password)
    {
        // Notice: Placeholders are used instead of the actual values.
        SqlCommand command = new SqlCommand(@"
            SELECT COUNT(*)
            FROM Users
            WHERE username = @username AND password = @password",
            _connection);
        /* Here, we "fill in" the placeholders with the correct values. The MSSQL driver
        * automatically escapes them so there's no chance of SQL injection vulnerabilities!
        * Alternative approaches include using stored procedures, or other database
        * technologies. Refer to my portfolio writeup for more information!
        */
        command.Parameters.AddWithValue("username", username);
        command.Parameters.AddWithValue("password", password);
        // Same check as previous method
        return (int) command.ExecuteScalar() != 0;
    }
}

```

```

C:\Users\Daniel\Documents\Wyy Dropbox\Unl\HIT3999 ... .NET\Portfolio\Code\SQLInjection\Direct\Program.cs 1
1 using System;
2 using System.Data.SqlClient;
3
4 namespace DirectSQLInjection
5 {
6     /// <summary>
7     /// Program to demonstrate SQL injection vulnerabilities and how they can be avoided
8     /// </summary>
9     <author email="daniel@dan.cx">Daniel Lo Nigro</author>
10    class Program
11    {
12        protected const string CONNECTION_STRING = @"
13        Data Source=DANIEL-LAPTOP2\SQLSERVER;Initial Catalog=Gadgets;
14        Integrated Security=True;Pooling=False";
15
16        protected static SqlConnection _connection;
17
18        static void Main(string[] args)
19        {
20            ConnectToDatabase();
21            ShowTypes();
22
23            do
24            {
25                Console.WriteLine("select type: ");
26                string type = Console.ReadLine();
27                VulnerableList(type);
28            } while (true);
29
30            Console.ReadKey();
31        }
32
33        protected static void ConnectToDatabase()
34        {
35            _connection = new SqlConnection(CONNECTION_STRING);
36            _connection.Open();
37        }
38
39        /// <summary>
40        /// Show a list of all the available gadget types
41        /// </summary>
42        protected static void ShowTypes()
43        {
44            Console.WriteLine("Available gadget types:");
45            SqlCommand command = new SqlCommand("SELECT GadgetTypeName FROM GadgetType", _connection);
46
47
48
49
50            using (SqlDataReader reader = command.ExecuteReader())
51            {
52                while (reader.Read())
53                {
54                    Console.WriteLine(reader["GadgetTypeName"]);
55                }
56            }
57            Console.WriteLine();
58
59
60            /// <summary>
61            /// Display all gadgets of a certain type. Vulnerable to SQL injection!
62            /// </summary>
63            <<param name="gadgetType"></param>
64            protected static void VulnerableList(string gadgetType)
65            {
66                SqlCommand command = new SqlCommand(
67                    @"
68                    @
69                    SELECT GadgetID, m.Name Manufacturer, g.Name
70                    FROM Gadget g
71                    LEFT OUTER JOIN GadgetType t ON t.GadgetTypeID = g.GadgetTypeID
72                    LEFT OUTER JOIN Manufacturer m ON m.ManufacturerID = g.Manufacturer
73                    WHERE t.GadgetTypeName = '{0}',
74                    gadgetType),

```

```

C:\Users\Daniel\Documents\Wyy Dropbox\Unl\HIT3999 ... .NET\Portfolio\Code\SQLInjection\Direct\Program.cs 2
75
76     _connection);
77     using (SqlDataReader dataReader = command.ExecuteReader())
78     {
79         while (dataReader.Read())
80         {
81             Console.WriteLine("{0} by {1} (ID = {2})",
82                 "GadgetID"]);
83         }
84     }
85     Console.WriteLine();
86
87     /// <summary>
88     /// Check username and password are valid. NOT vulnerable to SQL injection!
89     /// </summary>
90     <<param name="username">The username.</param>
91     <<param name="password">The password.</param>
92     <<returns>True if user is valid, otherwise false</returns>
93     protected static bool FixedCheck(string username, string password)
94     {
95         // Notice: Placeholders are used instead of the actual values.
96         SqlCommand command = new SqlCommand(@"
97         SELECT COUNT(*)
98         FROM Users
99         WHERE username = @username AND password = @password",
100            _connection);
101         /* Here, we "fill in" the placeholders with the correct values. The MSSQL driver
102            * automatically escapes them so there's no chance of SQL injection vulnerabilities!
103            * Alternative approaches include using stored procedures, or other database
104            * technologies. Refer to my portfolio writeup for more information!
105            */
106         command.Parameters.AddWithValue("username", username);
107         command.Parameters.AddWithValue("password", password);
108         // Same check as previous method
109         return (int)command.ExecuteScalar() != 0;
110     }
111 }
112
113 }
114

```


Security Concerns for Web Services

Short Report

Security Concerns for Web Services

1 Introduction

Security is a major concern for any public-facing system. Web services are especially important to secure, as often they power the backend for several systems. This short report will focus on the various security aspects that require consideration when developing web services. Unless otherwise stated, this short report will focus exclusively on SOAP-based web services.

There are six main security concepts that need to be focused on when doing security testing (Wikipedia 2011b). They are:

2 Authentication

Authentication is verifying that the user accessing the web service is allowed to do so. Most of the time, this is done by verifying user credentials of some form (although, for some web services only used internally by a company, this can be done by IP-based restrictions). Depending on how secure the web service needs to be, there are several different methods to do this

2.1 Basic Username / Password

A large number of web services use a normal username and password for authentication the user. Generally, this should only be done over SSL (via HTTPS), otherwise someone sniffing the connection could steal the password. The WS-Security standard specifies a standard method of doing this – A custom envelope header that is added to every message (Nadalin et al. 2006). This is very similar to how Basic HTTP authentication works – Basic authentication adds a HTTP header with a Base64-encoded version of the username and password.

If added security is required (or SSL can't be used), digest authentication can be used. Unlike basic authentication, digest authentication uses encryption instead of sending the password in plaintext (Wikipedia 2011a). This is also supported by the WS-Security standard (Seely 2002).

2.2 Client certificate

For situations that require added security, a client certificate (also known as an X.509 certificate) can be used. Essentially, this is similar to how SSL works, except it is the other way around. The client sends its certificate to the server, which validates that it's a valid certificate, and the certificate owner is able to access the server (Wikipedia 2011c).

Generally, this is done using SSL (i.e. HTTPS) as a transport. The client verifies the server is who is claims to be (as with a normal SSL connection), and the server verifies the client in the same way. This is often known as "Mutually Authenticated SSL" or "Mutual authentication".

2.3 Others

There are many other methods of authentication, including Kerberos tickets and custom authentication tokens. I will not discuss these in detail as they are not widely used compared to the other two methods.

3 Authorization

Authorization, while often considered a part of authentication, is a separate issue. While authentication is the act of validating that the user is recognised by the system, authorization is the act of validating that they're **allowed** to do what they're trying to do (i.e. whether they have the right to access the service or data).

Most of the time, this is implemented using some role-based or group-based system. For example, normal users should be able to read data, whereas super-users or administrators are usually allowed to update or create data. The ASP.NET membership provider supports this via roles, and is extensible so you can use your own database with it.

4 Confidentiality

Confidentiality is the guarantee that nobody else can “eavesdrop” on the conversation between the client and the server. This prevents “man in the middle” attacks, where the client is connecting to a third party masquerading as the proper server, proxying all the data (Geuer-Pollmann & Claessens 2005). One common method to ensure confidentiality is to use network layer security such as SSL or some sort of mutual authentication, as mentioned in the “client certificate” section above.

If this is not possible, message-level encryption can be used. Message-level encryption is basically encryption of the payloads of SOAP calls. The client and server both need to agree on the encryption type to use, and potentially have a shared encryption key (shared before the web service calls).

The difference between transport-level encryption and message-level encryption is that the application doesn't really need to worry about transport-level security, as it is provided by a lower layer. With message-level encryption, the application needs to manually encrypt all the messages.

Transport-level security can often be added as an enhancement to existing web services, as it generally does not require any changes to the code. Message-level encryption will almost **always** require code changes, as it is done at the application layer.

5 Integrity

Integrity is the guarantee that the message was not modified in its journey from client to server. This is similar to confidentiality, and related in some ways. The most common way to ensure integrity is to **sign** messages. Signing is very similar to encryption in the way that it works. However, instead of encrypting the whole message, signing involves getting a hash of the message, using the client's **private key**. The server can use the client's **public key** to check that the signature is authentic. Because any changes to the message would cause a change to the signature, and the signature can only be generated using the client's private key, this ensures that the message was not changed in transport.

6 Non-repudiation

Non-repudiation is the guarantee that the sender of the message can't deny that they sent it at a later point in time. Generally, this is achieved by having an audit trail (logging every web service call) so that all the changes made can be proven later. Quite often, this goes hand-in-hand with integrity

– Messages are signed (so that you are sure they haven't been tampered with) and then logged (to keep as future proof that the messages were not tampered with).

7 Accessibility

Accessibility is probably the one topic people think of when they think of security. Accessibility is ensuring that the system is always accessible, and is not impaired by DoS or DDoS attacks.

Denial of Service attacks (otherwise known as “DoS attacks”) are a common issue for all modern web applications and web services. A denial of service attack is when one client sends you hundreds or even thousands of rogue hits per second, with the intention of overloading your system and causing it to go offline for everyone (denying them access to the service).

Denial of service attacks can often be blocked at a firewall level, by analysis of traffic. If it is receiving a large number of similar hits from one IP address, block that IP address from accessing the server (Kargl, Maier & Weber 2001). If it's not possible to do this at a firewall level, it can be done at an application level (although this involves more work). To do this, you'd save the number of hits received from each IP address into a database. If the number of hits exceeds a specific threshold, block the IP for a certain period of time.

Unfortunately, DDoS (Distributed Denial of Service) attacks are often very hard to mitigate. The difference between DoS and DDoS attacks is that while a DoS attack involves a single client, a DDoS attack involves many (often tens or even hundreds) of clients. These clients are often part of botnets, which are networks of computers infected with viruses that all report to one central “control” server (Mirkovic & Reiher 2004). The control server tells the botnet clients where to attack, and they all attack it.

Because it is distributed, it is often hard to tell the difference between a DDoS attack, and a large amount of legitimate traffic in a short period of time. It may be possible to block DDoS attacks if all the requests have a pattern. For example, they all request the same specific URL, you can block access to that URL.

8 Summary

Security is a major concern for any public-facing system, and web services are no different. Since they often provide the backend for several systems, any security issues in web services are relatively high-risk. Focusing on the six main security concepts (authentication, authorization, confidentiality, integrity, non-repudiation and accessibility) will help ensure the security of your web service.

9 References

Geuer-Pollmann, C & Claessens, J 2005, 'Web services and web service security standards', *Information Security Technical Report*, vol. 10, no. 1, pp. 15-24,

Kargl, F, Maier, J & Weber, M 2001, 'Protecting web servers from distributed denial of service attacks,' *ACM*, 514-524.

Security Concerns for Web Services

Mirkovic, J & Reiher, P 2004, 'A taxonomy of DDoS attack and DDoS defense mechanisms', *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39-53,

Nadalin, A, Kaler, C, Monzillo, R & Hallam-Baker, P 2006, *Web services security: Soap message security 1.1 (ws-security 2004)*, viewed

Seely, S 2002, *Understanding WS-Security*, viewed 2011-06-02, <<http://msdn.microsoft.com/en-us/library/ms977327.aspx>>.

Wikipedia 2011a, *Digest access authentication - Wikipedia, The Free Encyclopedia*, viewed 2011-06-02, <http://en.wikipedia.org/wiki/Digest_access_authentication>.

Wikipedia 2011b, *Security testing - Wikipedia, The Free Encyclopedia*, viewed 2011-06-02, <http://en.wikipedia.org/wiki/Security_testing>.

Wikipedia 2011c, *X.509 - Wikipedia, The Free Encyclopedia*, viewed 2011-06-01, <<http://en.wikipedia.org/wiki/X.509>>.

ASP.NET Web Forms

Short report

ASP.NET Web Forms – Short Report

1 Introduction

ASP.NET web forms was (and still is) the “classic” method of creating sites using ASP.NET technology. It was designed to mimic windows forms in a web-based environment, including the event model (Liberty & Hurwitz 2003). This short report will briefly talk about the advantages and disadvantages of the ASP.NET web forms model.

2 Advantages

Most (if not all) of the advantages listed below are the advantages ASP.NET Web Forms have over “classic ASP” and other interpreted languages. Developers exclusively using Microsoft technologies would have encountered classic ASP in the past.

2.1 Code-Behind Model

Previously (with classic ASP), business logic and presentational code could be combined in the one file. This often resulted in spaghetti code and tightly coupled code that is very hard to modify. It was possible to create a “framework” to handle the separation of business logic and presentational code, but this was rarely done.

The “code-behind” model was originally introduced as part of ASP.NET 1.0, and significantly enhanced in ASP.NET 2.0 (Patel, Acker & McGovern 2006). In this model, controls and static HTML are placed into a frontend file (.aspx), and all the code to interface with the controls is placed in a code-behind file (.aspx.cs for C# and .aspx.vb for VB.NET). This is an implementation of the “separation of concerns” principle – The UI code goes in one file, and the code that does all the work goes in another file. This is a significant improvement over the previous ASP (Active Server Pages) scripting framework, where business logic and presentation code were combined into the one file.

2.2 Compiled = Faster

Traditionally, web scripting languages were interpreted languages such as VBScript or JScript (Microsoft’s version of JavaScript) as used in ASP. These languages are not compiled; instead they’re interpreted every time you go to the page. For every page hit, the scripting language runtime needs to load the file, parse it into an abstract syntax tree, and then execute the contained commands. Doing this every hit causes noticeable performance degradation.

On the other hand, ASP.NET code (like all .NET code) is compiled. The code-behind files can be compiled before deployment, and the .aspx files are compiled at runtime by IIS, the first time the site is hit. Both are first compiled into Common Intermediate Language (CIL) code by the compiler, and are then compiled into machine code at runtime by a “Just in Time” (JIT) compiler.

This means that ASP.NET has performance advantages over older technologies such as ASP (Active Server Pages) which are interpreted. The interpretation of the .aspx files only happens once (on the first hit), meaning that loading the site should theoretically be faster than previous technologies. Compiling CIL code into machine code is significantly faster than interpreting a whole file.

2.3 Rapid Application Development

Rapid development of applications in ASP.NET is definitely possible, and is one of its strongpoints (Liberty & Hurwitz 2003). Applications can be developed quickly and easily simply by dragging pre-built (and custom) controls on to the page, and data-binding them to a business layer component.

Due to its popularity, there is a very wide range of third-party components available for ASP.NET web forms, including advanced controls like spreadsheet and graphing components. Companies such as Telerik are dedicated to creating controls for ASP.NET.

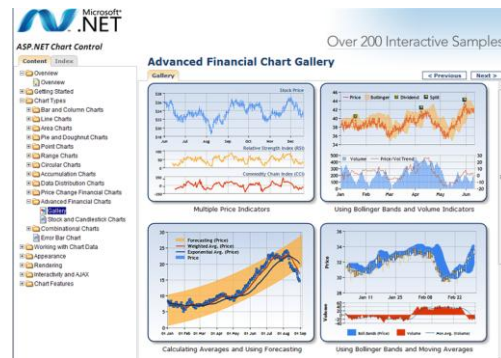


Figure 1: Example of a pre-built charting control for ASP.NET Web Forms.

2.4 Easy Development Model for WinForms Developers

The ASP.NET Web Forms model is based off the classic Windows Forms model. That is, it is event driven. For example, whenever you click on a button, a button click event is fired on the server. Developers with previous Windows forms experience should feel right at home, as a lot of the abstractions (such as events and event listeners) are the same. Familiar events such as Click, KeyPress and SelectedIndexChanged (for lists) are all supported using similar handlers.

The ASP.NET framework handles all the form posts and form data required to make the event handlers work. Since HTML is stateless, the current “state” of the page needs to be stored in order to make the event system work, which is often a cause of frustration for ASP.NET developers (see “disadvantages” section below).

3 Disadvantages

The ASP.NET web forms model is powerful and has a number of advantages over previous technologies, but it does have some downsides. Many of these relate to the control and event abstractions involved in the framework.

3.1 Unit-Testing Impossible

One of the downsides of ASP.NET web forms is that unit testing web forms pages is close to impossible, because you can't instantiate new instances of pages without going through the whole (rather complex) ASP.NET Page Life Cycle (Microsoft). ASP.NET web form applications are often tightly coupled to all controls used by the page, which makes the code quite un reusable and untestable, and increases coupling.

Controllers in the ASP.NET MVC framework are unit testable as they can be instantiated very easily. ASP.NET MVC doesn't have the complicated page life cycle to worry about.

3.2 Autogenerated HTML

The controls abstraction used in ASP.NET web forms may be great in theory, but it leads to HTML being automatically generated. This HTML is often bloated with many levels of tables (although this has been partially resolved in newer ASP.NET versions) and is semantically ugly. Using CSS to style

the output of a lot of controls is often a pain to do, due to the unsemantic code. Also, you lack control over the HTML output, as you’re not actually writing any of it yourself.

3.3 Viewstate Bloat

HTTP is stateless. This means that each request is totally separate from all previous requests, and the server does not keep track of all the requests. Once a request is completed, the server totally “forgets” about it, and the next request to the same page is fresh. Most web development frameworks solve this by using sessions and databases to persist data between pages. Often, details such as the currently logged in user are stored in a session variable on the server, so it persists across pages.

The statelessness of HTTP poses a problem for the ASP.NET web forms model – Since it uses an event-driven design, it needs to know what has changed (i.e. if the contents of any text boxes has changed, if the state of any checkboxes has changed, etc.). This can’t really be stored in a session variable, as it’s unique per page load.

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="jXhw87aEh0
+huDnn1o1nP55QLpVtX03AqBxozGkA1HSD0Hovou8Y9GPrR5kX51IX9q5a61uB3NnTlM17arxw6Flac4ETwZqAKuK3Kfkn+8HQ
VlXrH9tze6A1e06145cQ801zXkY1bw85PQRvrvuNFJ3cdU1Lrj fea5H05WIDK+RlhgcER1ZLdyZXI9gINTE35E4y0016ccNBtDMLC
1PzNa/c1mL5P3t/nlqtU4uwxseuYm50aqqPcAhpPLSpfdor2aKxvIvrrpJ0E8AZG5DvZnShx/7EK0d09njUpxo0Pd78oboi sPFR
ZcF6cTN54V3sb+vd0eWw188vpoZfPdqvBx9A7zPhyt/5MlKtUuXxAWPr7akLPYU8GouLqj3NLxw+StbkOgmZLz6YH+n6ny9LIWu
DZke1n7/vyZ5xyjxUlcucQ603pLInnd0EKXfEk45E0kNyosDC6Ua1vLvuVrud3dFLF/QYZ2)g5rH1wxk61d77Kgt40Ordj3ZISG
qq0wMD140bEKJ0P8mC1VvYp1XZAdotLtu5Q8U8Jf2w0hAKPvCrBhpVTrLweJk8+
115FP75xvXQ8Z8WkxkfuwhX03AqBxozGkA1HSD0Hovou8Y9GPrR5kX51IX9q5a61uB3NnTlM17arxw6Flac4ETwZqAKuK3Kfkn+8HQ
USQ8A51/5aEBxThopu90x74RgrK950LzL0d8fzqnvYzdk2Y1azSrxgW1EekN3vww9hE2sgvMzI7KTyY110hpu33M0bz
CtnrOa1f0ndqN5gaet7s9WHEB1Uof/run1s/yTG7uH0Irs2BnhhdWQztznASXpNLpVnLxIFh04Cub2DN055E1/LK5Gnb3jE1j+
5ptPyeEEJEv+qioNf/w0wAnkH5KPDHvZSX9EZTuxvfuFuNa3+/PnAk1VKF5Kw4/uhvfy1oh9fCsn0qcnw9Krmg5y8Vgup78zVH
081IKNBvtU2WASoAKRC2dP+XkIar259D4V5t+Uf a9WPbz1+IFC372Qhqv3bgChcxidqALfaVQn9926e6S064Ez3o5WavQt174
XUuH1q3Mkr/j5DP1Qm80FFBxtt6QTyf9
+G1f0P8yCg743d3e9eaen1ER14FHswabY5G0HwCpwAzqno+g+emo7VneYfY7m0xc3z3cJQPHSTVpcc06m1Tawj1k1k1q
ab0u4890omof1hnF7zF8r4XRvR5r2N4H5JQd1T1d0Xeam91LEaqJ+TkswdgpcUx1pXAnuv56HrHD+QDP631YyIvncEv9g7Wd1Z
za1ZHV0uALhp3QkgluVUP5RRvGE27e1Pfwes68sgUeLz0ZgdR0mX09/vP2NzC1Oy4L8zrB56LInog6d1E3Qp5/op13ao2Ej8cQ
80PN/itIULPnFALmh10d1FAlpYsF4H2ekfPnPLdUnk13G+FWA6Dv6jvolfvceD6937aHPo5vs3vbd04C1cbqq911tdb3nbQNA/
8rcr1iGtausVg2f+5KfAYNBdpsT3+O/EQJc9LPTv519+r/TazrVen6ozYYfUMN2Gc1Go0Vus7a1o1B/
+Pz1kTY53h3c3HcIEAgyPCZ2o/Q1V/x/yca4A/mZf01eu0tEbVz8NULlgy/07QyWg4jNDqzgL0o56hV2x4jKgfG9Eopgh4Z1
zAKcR5QcPhy558r+Rl+j3h0FDom1ks3Domu6jUE1ML1Nbyb290hKL12myf6Pn0h55G5XrP9K1EzFaw58FkqDvNzX0G9jAnwQPNe
6u/9a0RwLcwr2Q9Yq9/14/LNK080m1s3d3jXtGf09s+cdceVFG1ZK75r5Nhw9Q6uXNRCPLQPes/1hFvUm2QWrmfMkXh+1p
```

Figure 2: Example of ViewState bloat

To solve this “problem”, the ASP.NET web forms framework stores the state of the current page in a hidden field called “ViewState”. When used incorrectly, the ViewState can grow to hundreds of kilobytes, causing significant delays to page load time.

ViewState bloat is often caused by developers not understanding how ViewState works, or using it incorrectly (Reed 2006). Without looking at the output HTML source of your web application, it’s very hard to tell that your ViewState is growing to unusable proportions. At work once, there was a page that kept timing out because it had a 10 MB viewstate¹.

If you are not using events in the page, viewstate can often be turned off, and this is the suggested best practice approach (Muhammad & Milner 2003). However, this can have unintended side effects (for example, dropdown lists **require** that ViewState is turned on).

3.4 Complicated Scenarios are Often Hard

Another disadvantage of ASP.NET web forms is that some more complicated scenarios (which are very easy to implement with other frameworks) are practically impossible to implement.

3.4.1 Dynamic creation of controls

One example is adding controls at runtime on the client side, via JavaScript. With other frameworks, you can simply insert the HTML required for the control (either created in JavaScript, or rendered server-side then returned via an AJAX call) and it all works nicely (as you’re just handling POST data). With ASP.NET web forms, since controls are built into a tree structure server-side, you can’t add more controls on the client side. Any added controls are ignored by the server. You can directly access the POST data to get the value of the control, but this bypasses the ASP.NET web form way of doing things (using “controls”).

¹ True story.

The ASP.NET AJAX toolkit added a control called an “UpdatePanel” that allows partial-page reloads (AJAX). Whilst this is an improvement (and does allow for dynamic adding of controls), UpdatePanel requests send all the standard POST data including the viewstate (Prosisie 2007) so they’re not very efficient at all. Additionally, they do not allow for control creation in JavaScript itself – Instead, the page section is rendered server-side and returned.

The ASP.NET MVC framework supports dynamic creation of controls, as it takes a different approach to controls. In the ASP.NET MVC framework, “controls” are just pieces of HTML, and POST data is handled directly.

3.5 “Ugly” URLs

A trend with modern web applications is to have “pretty”, descriptive URLs, often using RESTful practices. This keeps URLs clean and human-friendly as well as SEO-friendly. Until recently, ASP.NET web forms did not support these “pretty” URLs natively at all. You had to either use a third-party component, or create a component for it yourself.

Support for URL routing was added in the .NET Framework 3.5 SP1 (Allen 2009), using the same routing engine that the ASP.NET MVC framework uses. However, it is still a component that sits on top of an architecture that was not designed for it, and hence may not work correctly all the time (for example, postbacks commonly have issues with routed URLs). On the other hand, ASP.NET MVC was built from the ground-up to support URL routing.

4 Summary

ASP.NET web forms is a very powerful framework, allowing you to easily and rapidly create web applications using a development style similar to that of Windows forms. Due to its popularity, there is a large library of third-party components available for Web Forms, making rapid application development very easy.

However, there are several disadvantages of the framework, including “ViewState bloat”, a lack of control over the rendered HTML, and “ugly” URLs. In my opinion, it’s a level of abstraction that’s too high for web development. Web developers should know how HTTP works (and the fact it’s stateless) and develop web applications in the same way as they’re done in every other programming language.

5 References

Allen, S 2009, *Routing with ASP.NET Web Forms*, viewed 2011-05-29, <<http://msdn.microsoft.com/en-us/magazine/dd347546.aspx>>.

Liberty, J & Hurwitz, D 2003, *Programming ASP.NET*, 2 edn., O'Reilly & Associates, Inc., Sebastopol, CA, USA.

Microsoft *ASP.NET Page Life Cycle Overview*, viewed 2011-05-15, <<http://msdn.microsoft.com/en-us/library/ms178472.aspx>>.

Muhammad, F & Milner, M 2003, *Real world ASP. NET best practices*, Apress.

ASP.NET Web Forms – Short Report

Patel, J, Acker, B & McGovern, R 2006, *Feature Changes in ASP.NET 2.0*, viewed 2011-05-30, <<http://msdn.microsoft.com/en-us/library/aa479401.aspx>>.

Prosize, J 2007, *UpdatePanel Tips and Tricks*, viewed 2011-06-05, <<http://msdn.microsoft.com/en-us/magazine/cc163413.aspx>>.

Reed, D 2006, *TRULY Understanding ViewState*, viewed 2011-05-29, <<http://weblogs.asp.net/infinitiesloop/archive/2006/08/03/Truly-Understanding-Viewstate.aspx>>.

The Model-View- Controller Pattern

Short Report

The Model-View-Controller Pattern

1 Introduction

An alternative structure to the classic three-tier architecture is the MVC design pattern. The MVC pattern was originally created by Trygve Reenskaug for Xerox PARC (1979), with a more detailed description in a paper called “Applications Programming in Smalltalk-80: How to use Model–View–Controller” (Burbeck 1992). Since then, the pattern has been expanded to cater for web applications, in addition to traditional desktop applications (Selfa, Carrillo & Del Rocio Boone 2006). This short report will focus exclusively on the Model-View-Controller pattern as it applies to web applications.

2 Components

In the MVC architecture, there are several different components. There are the main components, which the name “MVC” is based off (models, views and controllers). In addition to these, web-based MVC frameworks often have other components such as a dispatcher.

2.1 Models

Models are basically the MVC equivalent of the data layer in traditional three-tier architectures. They’re responsible for database operations, as well as containing data from the database.

The same technologies used for data layers (such as ADO.NET) can be used to create the models, but ORM systems are used almost all of the time. This is usually either Entity Framework or NHibernate (or one of its variants such as Fluent NHibernate, Castle ActiveRecord, etc), as these are the most popular ORM systems in the .NET community.

When using an ORM system, the models would consist of the actual ORM entities themselves, as well as some CRUD methods. The CRUD methods are generally either part of a repository (if using the repository pattern), or static methods on the ORM entities (if using the Active Record pattern).

2.2 Views

Views are roughly the equivalent of a user interface layer. However, one of the major differences is that views are “dumb” – They do not call **any** business logic code at all! Logic in views is restricted to **only** view-related logic (for example, iterating to show more than one row, and showing “No results were found” only when a search returns no results). Views get passed model data from the

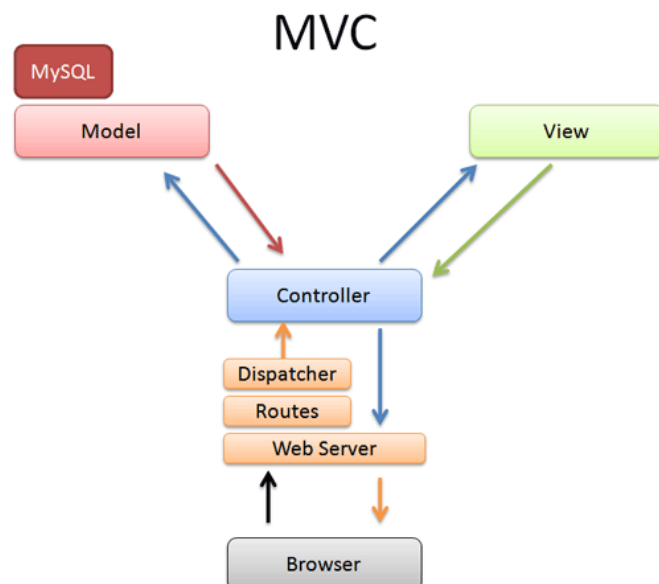


Figure 1: Diagram of the MVC pattern (Azad 2007)

controllers, which they render. This is their whole purpose in life. In a way, views are a little like the .aspx files used with ASP.NET web forms.

Views do not necessarily have to be HTML. For example, if you are using AJAX (and what modern sites don't?), you might have some views for JSON or XML results. Views can also be used internally – If your site sends emails, the email content would be stored in views.

2.3 Controllers

A controller is a class that handles incoming requests. One controller has multiple methods (referred to as “actions” in MVC terminology). The controller will generally load required data via the models, and then pass the data to the views to render. They may either implement the business logic themselves, or call business layer functions to do so. In most situations, there is one controller per business object (for example, a blog system may have a Post controller, a Comment controller and an Author controller).

In a way, controllers are somewhat similar to the code-behind files of an ASP.NET web forms application. However, a key difference between controllers and code-behind files are that controllers are not coupled to the views in any way. In a properly designed MVC application, you can totally replace the views without modifying the controllers at all (and vice versa). This also allows multiple controllers to use the same views, respecting the DRY (Don't Repeat Yourself) principle.

2.4 Dispatcher

Also known as a “router”, the dispatcher is generally a component that is built-in to the framework you use. Since it deals with URLs, it is specific to web-based MVC frameworks. Its role is to parse the URLs of incoming requests, and map them to specific controller actions. This is generally done by using a URL routing table. A routing table usually contains a mapping of URL patterns to controllers and actions. In the Kohana PHP framework, the routing table is setup in a “bootstrap.php” file¹. In the ASP.NET MVC Framework, the routing table is set up in the Global.asax.cs or Global.asax.vb file.

The default URL routing system in MVC frameworks is generally to use a URL structure like **controller/[action]/[id]**. With this default URL structure, URLs are routed something like this:

URL	Controller	Action	ID parameter
product/	Product	Default	Null
product/view	Product	View	Null
product/view/123	Product	View	123

This routing structure is fine for most applications, in that it works. Even with these default URLs, they still look nicer than the URLs generally used with traditional web applications (that use a programming language like ASP.NET or PHP, without an MVC framework).

However, most applications will probably want to use a custom URL routing structure, in addition to the default structure. This could potentially make the URLs more search-engine (or user) friendly. For example, with a product site you might want to include the product name in the URL, and with a blog site you'd generally want to include the date of the blog post in the URL (for example, /

¹ See <https://github.com/Daniel15/Website/blob/master/application/bootstrap.php#L134> for an example (my site's bootstrap code)

2011/05/15/new-blog-system). Most MVC frameworks are quite flexible and let you specify literally any URL routing scheme.

3 Advantages

There are several advantages of using the MVC pattern to design web applications. Some of the main ones are:

- **Separation of concerns** – If you follow it strictly, the MVC pattern forces you to put all your data access code into models, and your HTML (or XML, whatever) output into views. While it's possible to have ugly spaghetti-code MVC applications, the MVC pattern (and most MVC frameworks) generally “push” you in the right direction.
- **Reusability** – As controllers and views are not tightly coupled (views are basically just “blobs” of HTML), views can easily be reused in other applications. The same for models – You can reuse your models in other applications quite easily. If you wanted to, you could put all your models in a separate assembly, and reuse them across several web applications.
- **Distributed Development** – Since the views, controllers and models are all totally separate, you can distribute development quite easily. Designers can work on the views, and separate groups of developers can work on the controllers and models at the same time. Since views are very simple code, it is easy for designers to edit them.
- **Clean URLs** – Generally, web applications written with the MVC pattern have nice URLs compared to those that aren't. As mentioned in the Dispatcher section above, URL routing is almost always a core feature of MVC frameworks.

There are other advantages, but these are the main ones. In addition, there are some benefits that are specific to ASP.NET MVC (including the lack of ViewState, and the fact the HTML is handwritten) which I've covered in the writeup about ASP.NET MVC in my main portfolio document.

4 Differences with three-tier architecture

The main difference between these two architectures is that while the three-tier architecture is linear, MVC is more triangular. The controllers communicate with models to get data, and then pass this data to the views. Views do **not** communicate with the controllers (as opposed to the three-tier architecture, where the UI layer **does** (and must) communicate with the business logic layer). This makes views more reusable (compared to a UI layer implementation), as they are not coupled with any components.

However, the two architectures do not need to always be separate, and can be used in conjunction with one another. MVC can be used as a layer on top of a three-tier architecture – The MVC controllers can call business logic functions, and if the data access layer uses the domain model pattern, the MVC application can use it as the models.

5 Summary

The MVC design pattern is a great way to design and implement loosely-coupled web applications. It promotes separation of concerns and loose coupling, allowing easy reuse of all the components.

Most MVC frameworks include a URL dispatcher/router allowing clean and friendly URLs to be used in your application. There are some differences between the three-tier architecture and the MVC architecture, but both can be combined in the same application.

I used the MVC design pattern to design my own website. The PHP source code is available from <https://github.com/Daniel15/Website>.

6 References

Azad, K 2007, *Intermediate Rails: Understanding Models, Views and Controllers*, viewed 2011-05-30, <<http://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/>>.

Burbeck, S 1992, *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)*, viewed 2011-05-30, <<http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html>>.

Reenskaug, T 1979, *MVC XEROX PARC 1978-79* viewed 2011-05-30, <<http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>>.

Selfa, DM, Carrillo, M & Del Rocio Boone, M 2006, 'A Database and Web Application Based on MVC Architecture', in *Proceedings of the 16th International Conference on Electronics, Communications and Computers*, IEEE Computer Society, Washington, DC, USA.

Pros and Cons of the ADO.NET Provider Model

Short Report

Pros and Cons of the ADO.NET Provider Model

1 Introduction

ADO.NET is a database abstraction layer provided with the .NET framework. It uses an adapter pattern, allowing you to easily switch between different database systems (e.g. from Microsoft SQL Server to MySQL) without changing any of your code. This short report will attempt to describe some of the pros and cons of using the ADO.NET provider model and the database factory methods provided by the framework.

2 Advantages

The ADO.NET Provider Model is very powerful and has a number of advantages. These advantages are the main reasons why it is used.

2.1 One Single Consistant API

The main advantage of using a database abstraction layer such as the ADO.NET provider model is that you do not need to learn a new API for each database system. Languages such as PHP used to have this issue – Each database system (eg. MySQL, PostgreSQL, SQL Server, SQLite, etc.) had their own separate API, each with their own functions. To switch from SQL Server to MySQL, you needed to learn how the new API works (and learn how it differs from the one you were previously using), and edit **all** of your database code to use the new API.

With a database abstraction layer, you can simply change a configuration setting, and can instantly switch to a different database system with no code changes (provided the database has been created). This gives you a major advantage and cuts development time when porting to a different platform. ADO.NET is a good example of a database abstraction layer, and PHP has also implemented one called PDO (PHP Data Objects)¹.

2.2 Database Agnostic

Another benefit of using such a model (well, technically, it's an extension of the first benefit) is that you do not have any "vendor lockdown". You are free to switch to any other database system at any time. If MySQL is discontinued tomorrow², you can simply switch to a different system. As mentioned in the "one single API" section above, there would be no code changes required to switch to a different database system. If your company gets bought by Microsoft and they want you to switch from MySQL to SQL Server, it's very easy to do so.

In theory, that is. The reality is that there are many issues that may be encountered when switching from one database technology to another. Some of these are listed in the "disadvantages" section below.

¹ <http://php.net/manual/en/book.pdo.php>

² Very doubtful, but it could happen one day!

3 Disadvantages / Issues

Unfortunately, due to its very nature, there are a number of issues with the ADO.NET Provider Model. Most of these stem around using the same SQL queries across multiple database systems.

3.1 SQL Dialects

Much like dialects of a spoken language such as English, SQL comes in multiple dialects. While most database management systems support at least SQL92, there are several things that aren't defined in the standards, such as paging (loading a certain "page" of data). These "gaps" in the standards are usually implemented in vendor-specific ways, meaning that each database system has its own method for doing what is essentially the same thing.

Paging is a classic example. With MySQL³ or PostgreSQL⁴, a query such as the following would be used to retrieve the first 10 records in from a table:

```
SELECT * FROM Table LIMIT 10
```

With Microsoft SQL Server, the query looks like the following:

```
SELECT TOP 10 * FROM Table
```

Very subtle difference, but it gets worse when you want a certain section of the records. With MySQL and PostgreSQL, this is done very simply. To retrieve 10 records starting at number 20 (i.e. records 20 to 30), you can run a query like this:

```
SELECT * FROM Table LIMIT 10 OFFSET 20
```

In Microsoft SQL Server, this is not as simple, as there is no direct equivalent. SQL Server 2005 added a ROW_NUMBER function that helps with paging, however, it still requires considerably more SQL than MySQL does (Coles 2008).

One method often used to solve this issue is to have a "query builder". Instead of directly writing SQL, you call methods on the query builder, and it builds up the SQL query for you. An example of a possible (very simple) interface for a query builder is something like the following:

```
var query = new SelectQuery();  
query.From = "Table";  
query.Offset = 20;  
query.RecordCount = 10;  
var rows = query.GetRecords();
```

GetRecords() would generate and execute the SQL, and return all the rows.

A query builder approach is the approach taken by the NHibernate Object-Relational Mapping system, as well as Microsoft's own Entity Framework. Entity Framework uses LINQ queries to build SQL, internally using a query builder to construct the SQL statement (Blakeley et al. 2006).

³ A very popular open-source RDBMS. <http://www.mysql.com/>

⁴ Another popular open-source RDBMS. <http://www.postgresql.com/>

3.2 SQL Query Optimisation

When writing queries that work on multiple database systems, you need to write your queries as simply as possible, so they're portable to other database systems. Unfortunately, using a generic query does not allow you to optimise it to the database system, as the SQL is identical for every database system.

When you run a query, it's compiled into a version that the database system can understand, a "query optimiser" is used to optimise this query, and a "query plan" is calculated for the database query. The query plan says exactly what the database system will do to execute the query (i.e. which indexes will be used, what data will be loaded, which joins will be done, etc.)

Every database system has its own unique query plan builder, and so different database systems optimise advanced queries in different ways. This means that a query that is very fast in SQL Server might be significantly slower on MySQL or PostgreSQL, and different methods of optimisation need to be used (index or locking hints, for example).

3.3 Database-specific Features

Database systems often come with plenty of built-in features. For example, SQL Server has a "PIVOT" feature that lets you "rotate" a table – that is, transform rows into columns (Cunningham, Galindo-Legaria & Graefe 2004). Most database systems also come with functions for manipulating timestamps. There are no set standards for these features and they vary from database system to database system, hence you can't really abstract their usage.

A common example is date handling functions. With MySQL, to get all the entries that were created in June, you can run a query similar to this:

```
SELECT * FROM Table WHERE MONTH(CreatedDate) = 6
```

However, this does not work with all DBMSes (for example, it doesn't exist in SQLite⁵). The MONTH function **does** exist in SQL Server, but other MySQL functions (eg. MONTHNAME) don't. Some functions do exist across different database systems, but have slightly different meanings or outputs.

Another example, to get all the entries that were created in the last 60 days, with MySQL you can run a query like the following:

```
SELECT * FROM Table WHERE CreatedDate > NOW() - INTERVAL 60 DAYS
```

This syntax does not exist in SQL Server.

Using the ADO.NET provider model and provider factory, you have two choices:

- Don't use DBMS-specific features, instead relying on standard SQL to do what you want. For example:
 - Instead of using time functions, work with standard UNIX timestamps stored as integers
 - Or, use DATETIME fields but do time calculations in your application manually. For example, to get the entries created in June, you could use standard SQL syntax:

⁵ A lightweight, embedded database system. <http://sqlite.org/>

```
SELECT * FROM Table WHERE CreatedDate BETWEEN '2011-06-01' AND '2011-06-30'
```

- Run different SQL statements depending on the database system in use. This ties your solution down to specific database systems, which is what the provider model aims to avoid.
 - Often results in ugly code like:
If (Microsoft SQL Server)
...
elseif (MySQL)
...

4 Summary and Conclusion

The ADO.NET Provider Model has a number of advantages. It's a single API that you can use to access a large number of database systems, meaning you do not need to learn a new API when switching to a different database system. However, this model is not without its flaws. Different SQL dialects and DBMS-specific features make it hard to create a full database abstraction at such a low level.

I believe that to use the ADO.NET Provider Model to its full potential, its usage needs to be combined with a "query builder" that abstracts the building of SQL queries. An example of this would be Microsoft's Entity Framework. ADO.NET would handle all the low-level details (the actual connection and sending/receiving of raw data), and the higher layer (Entity Framework or NHibernate) would handle the actual query generation.

However, in performance-critical situations, it may be necessary to optimise the queries specifically for the database system in use. In cases like this, it is best to stick to a single database system and optimise your code specifically for that system.

5 References

Blakeley, JA, Campbell, D, Muralidhar, S & Nori, A 2006, 'The ADO.NET entity framework: making the conceptual level real', *SIGMOD Rec.*, vol. 35, no. 4, pp. 32-39,

Coles, M 2008, 'Common Table Expressions and Windowing Functions', in *Pro T-SQL 2008 Programmer's Guide*, Apress, pp. 247-272.

Cunningham, C, Galindo-Legaria, CA & Graefe, G 2004, 'PIVOT and UNPIVOT: optimization and execution strategies in an RDBMS', *VLDB '04*, pp. 998-1009,

Enterprise .NET Portfolio

Miscellaneous Stuff

1 Table of Contents

2	Assignment 1.....	4
2.1	Task 1: Programming Glossary.....	4
2.1.1	Boxing.....	4
2.1.2	Generics	4
2.1.3	Iterators / Yield	4
2.1.4	Extension Methods	5
2.1.5	Collection Initialisers.....	5
2.1.6	Lambda Expressions.....	6
2.2	Concept map.....	6
3	Three-Tier Architecture.....	7
3.1	Data Access Layer (Assignment 3)	7
3.1.1	ADO.NET.....	7
3.1.2	LINQ to SQL	8
3.1.3	Entity Framework.....	8
3.1.4	Differences between classic and newer – Assignment 3 Task 4.....	9
3.1.5	Repository Pattern	9
3.2	Business Logic Layer (Assignment 6).....	9
3.2.1	Table module	10
3.2.2	Transaction scripts	11
3.2.3	Domain model.....	11
3.2.4	Service layer	12
3.3	User Interface Layer.....	12
3.3.1	Client-side application.....	12
3.3.2	Web-based	13
4	Web Services.....	14
4.1	Protocols	14
4.1.1	SOAP.....	14
4.1.2	REST.....	15
4.2	.NET Technologies.....	16
4.2.1	ASMX Web Services	16
4.2.2	Windows Communication Foundation (WCF)	16
4.3	Security concerns.....	17
5	COM+ (Assignment 4 Task 2)	17

Other Portfolio Stuff

6	MSMQ (Microsoft Message Queues).....	17
6.1	Advantages.....	17
7	Windows Services	17
7.1	Issues.....	18
7.1.1	Showing progress.....	18
7.1.2	Error handling	18
8	References	19

2 Assignment 1

2.1 Task 1: Programming Glossary

2.1.1 Boxing

In C#, there are two different types – Reference types, and value types. Reference types, such as most classes (including strings), are stored on the managed heap and passed around by value, whereas value types, such as integers, booleans and structs, are passed by value and stored on the stack. Boxing is the process of converting a value type into a reference type, by “wrapping” it inside a System.Object (Microsoft 2010). Boxing is automatically done when using a value type as an object (for example, for a parameter that accepts any object), whereas unboxing is explicit (you need to do it yourself).

For example, this code will cause the integer “testInt” to be boxed:

```
int testInt = 42;
object testObj = testInt;
```

And this will unbox the integer (notice the explicit cast):

```
object testObj = 42;
int testInt = (int) testObj;
```

Previously, collections such as Dictionary and ArrayList needed to box/unbox integers to store them. This issue was resolved with the introduction of generics (see next section)

2.1.2 Generics

Generics, added in .NET Framework 2.0, are basically .NET’s implementation of type parameters (similar to templates in C++, and generics in Java). They allow you to create classes that take type names as parameters, maximising code reuse and type safety. Generics are commonly used to implement collection classes (see the System.Collections.Generic namespace for all the generic collections that come built-in with the .NET Framework).

See the attached code for an example of a **Stack** implemented using generics (SimpleStack.cs)

2.1.3 Iterators / Yield

An enumerator is the magic that allows the **foreach** loop to work. Essentially, it tracks the current state of the iteration, and allows you to move forward through the list. To implement an enumerator, you need to implement the IEnumerable interface, which include a “Current” property (returns the current item), and a MoveNext method (to move to the next item).

An iterator is an easy way to create an enumerator for a class. Instead of having to create a totally separate IEnumerable implementation, the compiler does it for you. All you need to do is use the “yield return” feature.

To add an iterator to the stack implementation from 2.2, a very small amount of code is needed:

```
public IEnumerable<T> GetEnumerator()
{
    while (HasItems)
    {
```

Other Portfolio Stuff

```
        yield return Pop();
    }
}
```

This is used in a standard foreach loop:

```
foreach (string item in stringStack)
{
    Console.WriteLine(item);
}
```

2.1.4 Extension Methods

Extension methods allow you to “add” methods to existing classes, without having to modify the type itself or create a derived type. Internally, extension methods are actually implemented as static methods. These are similar to “mixins” in other languages (such as JavaScript), allowing you to add new functionality to existing classes.

A simple example is adding an “IsEmail” method that checks if a string is a valid email address:

```
// Regex from http://www.regular-expressions.info/email.html
private static readonly Regex _emailRegex = new Regex(@"\b[A-Z0-9._%~]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b",
    RegexOptions.Compiled | RegexOptions.IgnoreCase);

public static bool IsEmail(this string str)
{
    return _emailRegex.IsMatch(str);
}
```

The “this” before the parameter type tells the compiler that this is an extension method. This method would be called like any other string method:

```
string email = "daniel@dan.cx";
if (email.IsEmail())
    Console.WriteLine("Good! :");
else
    Console.WriteLine("Bad :(");
```

2.1.5 Collection Initialisers

A collection initialiser allows you to easily initialise a collection at runtime, using a simplified syntax. This saves you having to call the Add method over and over again.

Instead of doing this:

```
IList<string> cities = new List<string>();
cities.Add("Melbourne");
cities.Add("Sydney");
cities.Add("Adelaide");
// ... and so on
```

You can simply do this:

```
List<string> cities = new List<string> {"Melbourne", "Sydney", "Adelaide", ...};
```

This is very similar to the array initialisation syntax in languages such as C and C++.

2.1.6 Lambda Expressions

Lambda expressions are essentially C#'s version of anonymous or inline functions. Lambda functions were added in the .NET Framework 3.5, and simplify the syntax required for delegates. Previously, delegates were required to be separate methods, but with C# 3.5 and higher, they can be inline.

Given the list from the “collection initialisers” section above, the following code could be used to retrieve all the cities beginning with the letter A:

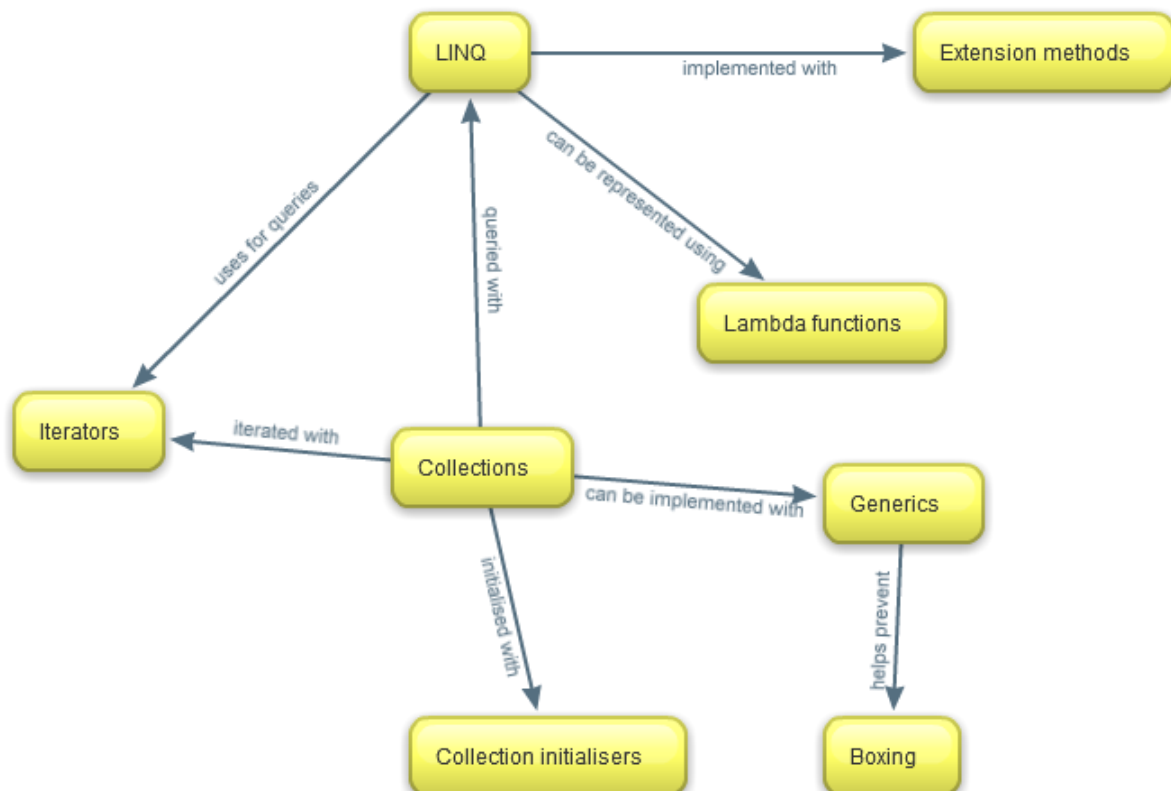
```
List<string> startsWithA = suburbs.FindAll(item => item.StartsWith("A"));  
foreach (string item in startsWithA)  
{  
    Console.WriteLine(item);  
}
```

LINQ uses many technologies including extension methods and lambda expressions in order to add a query language into the .NET Framework. It lets you query anything (databases, XML files, collections) using a language similar to SQL (or, alternatively, with normal method syntax). The FindAll line above could be replaced with the following to make use of LINQ:

```
var startsWithA = suburbs.Where(item => item.StartsWith("A"));
```

2.2 Concept map

The concept map below shows how all the above concepts are related:



3 Three-Tier Architecture

Most business .NET applications use some sort of **three-tier architecture**. This splits a system into a number of distinct layers – Data Access Layer, Business Logic Layer, and User interface layer. Layers are restricted to which other layers they can talk to – They can only talk to the layer directly below them, and themselves. The User Interface layer can only talk to the business logic layer, and the business logic layer can only talk to the User interface layer.

An alternate design pattern is MVC (Model-View-Controller), which I talk about in a short report. MVC follows a lot of the same principles, but goes about them in a slightly different way.

3.1 Data Access Layer (Assignment 3)

The data access layer generally connects to a database of some sort, and performs CRUD (Create, Read, Update and Delete) actions. However, the DAL is not database-specific – The data could come from anywhere (eg. a webservice that accesses a third-party system). Code that interacts with a third-party system of some sort would generally be kept in a data access layer.

There are a number of database technologies that are commonly used by .NET developers to interact with databases. Some of the most common are listed below:

3.1.1 ADO.NET

ADO.NET is the “classic” method of accessing databases from .NET. It was designed to be an evolution of ADO (ActiveX Data Objects), which was the technology used for database access in ASP. It uses a table module approach (via DataSets) to accessing the database. Even if not used directly, it’s generally used as the backend for every other .NET database access technology, as it’s a proven technology that comes built-in with every .NET version.

ADO.NET uses an adapter pattern for interfacing with different database systems in a consistent way. As I explain later, there are both advantages and disadvantages of this approach.

A technology called LINQ to DataSets is available, but this is generally only used to filter returned DataSets. For using LINQ to query a database, a technology such as LINQ to SQL or Entity Framework should be used. These are discussed in sections 3.1.2 and 3.1.3 respectively.

Pros:

- Available in every .NET Framework version
- Supports a large number of DBMSes (any that have an adapter available), including Microsoft SQL Server and variants (Express, CE), SQLite, MySQL, Oracle and many more.

Cons:

- May be harder to use than newer technologies. ADO.NET is not as “refined” as newer technologies.
- Need to write queries manually (although this can be a benefit, because you can optimise them)

3.1.2 LINQ to SQL

LINQ to SQL is a technology that lets you use LINQ queries to query a database. It was the first object-relational system created by Microsoft for the .NET Framework. It is relatively easy to use, as it uses a drag-and-drop designer in Visual Studio to create the classes, and normal LINQ queries to query the database. It supports lazy loading (only loading data when required), which is turned on by default.

Pros:

- Relatively easy to use
- Drag and drop UI in Visual Studio
- Lazy loading supported

Cons:

- Only supports Microsoft SQL Server and variants (SQL Server Express, SQL Server CE)
- Uncertain future? Entity Framework seems to be the way of the future and has more advantages, so development of LINQ to SQL is uncertain
- Only supports a 1-to-1 mapping between the database and the classes

3.1.3 Entity Framework

Entity Framework is the latest data access technology from Microsoft. Similar to LINQ to SQL, it lets you use LINQ queries to query the database, instead of having to write SQL (Blakeley et al. 2006). It supports complex models, including inheritance to model “is-a” relationships (Thompson 2007). Unlike LINQ to SQL, it is built on top of the ADO.NET adapter model, meaning that it supports all database providers that ADO.NET supports.

Entity Framework 4.1 has added a feature called “Entity First” – This lets you create your POCOs (Plain old CLR/C# Objects) and map these, instead of having to use automatically generated classes. It also lets you create a whole database schema based on the mapping you create. This means you don’t need to write any DDL (Data Definition Language) code, you just need to create your C# classes, and the database will be generated for you.

Mapping in the Entity Framework is done on three levels:

1. A conceptual model (CSDL file) – This maps all the entities that are used in the code itself. This models how you “see” the database in code.
2. A storage model (SSDL) – This contains a one-to-one mapping of the underlying database structure. This models how the database exactly.
3. A mapping (MSDL file) – This maps the conceptual model to the storage model. This is used to map how the data is stored to how the data is used.

Generally, these three files are combined together into a single Entity Data Model (EDMX) file.

Pros:

- Supports large number of DBMSes (same as ADO.NET)
- Very powerful

Other Portfolio Stuff

- Very flexible – The mapping system gives the framework a large amount of flexibility.
- Relatively easy to get started with

Cons:

- Complex – Due to the complexity of the mapping system, using the Entity Framework is a lot harder than LINQ to SQL.

3.1.4 Differences between classic and newer – Assignment 3 Task 4

The main difference between newer technologies (such as LINQ to SQL and Entity Framework) and older technologies like DataReaders and DataSets is that with the older technologies, you are generally required to build the SQL queries yourself. These could either be done as stored procedures, or as normal SQL statements. With LINQ to SQL and Entity Framework, the SQL queries are built automatically by the framework.

Because they are automatically generated, queries generated with LINQ to SQL and Entity Framework can have several issues. One of the most common issues is the “SELECT N+1” problem¹ – When joins are done incorrectly, significantly more queries are done than are needed (one for each sub-object). This problem is generally not possible when using raw ADO.NET, as you write your queries manually.

Additionally, since raw ADO.NET is a lower level of abstraction, it’s generally faster than using a higher level of abstraction such as Entity Framework or LINQ to SQL (Hanselman 2010). Small code that is very performance critical is probably best off using a DataReader, but for most uses, Entity Framework is quite powerful and handy.

3.1.5 Repository Pattern

While not a database access technology, the repository pattern is often a very important pattern to use when designing a data access layer, especially when using the domain model pattern (described in more detail later on). A repository is basically a class that handles CRUD operations for a certain table. This allows you to keep data access code separate from the entities themselves. The opposite would be the Active Record pattern, which has the CRUD operations as methods on the entities themselves.

The repository pattern makes it very easy to create “mock” objects for unit testing. A mock object is an object that simulates functionality of a normal object. When doing unit tests, you do not want to use an actual database, as the data could be different every time. Instead, you use mock objects that return fake data (and handle inserts and deletes internally). If all your data access code is contained in one section (repository classes), it’s very easy to create mocks for the data.

3.2 Business Logic Layer (Assignment 6)

The business logic layer is responsible for handling all business logic, which includes enforcing business rules, and calling the required methods in the data access layer. In other words, the business logic layer is all the code that makes the application actually work (otherwise the application would just be a simple UI on top of a database). Business rules are basically constraints

¹ See <http://stackoverflow.com/questions/97197/what-is-the-n1-selects-problem> for more information

that exist in your business model. For example, if you were writing an online shopping system, you may have business rules such as:

- Items can only be purchased if they are in stock
- If an item is on special, customers are limited to a quantity of 5

These rules are all enforced in the business logic layer. Generally, if any of the rules are violated, an exception is thrown (which can then be caught by the higher layer, the User Interface layer). If all the business rules are satisfied, the business layer uses the data access layer to persist any state changes to the database.

The business layer is designed so it's independent of both the layer above (the UI layer) and the layer below (the Data Access Layer). Either layer should be able to be swapped out and replaced with an alternative implementation, with no issues in the business logic layer. And the business layer should certainly NOT have any UI or database code in it! This makes it highly reusable, and increases cohesion.

As with the data access layer, there are quite a number of design patterns that can be used to create a business logic layer. In his book, Martin Fowler describes some of the most common patterns used for creating business logic layers (2002). I've summarised the most common patterns below.

3.2.1 Table module

The table module pattern organises the domain with one class per table in the database. These classes act as "managers" for the tables – A single instance of the class is used to perform CRUD operations to the table (compare this to the domain model pattern in section 3.2.3, which uses one object per entity in the database). Data is generally returned in the form of DataSets.

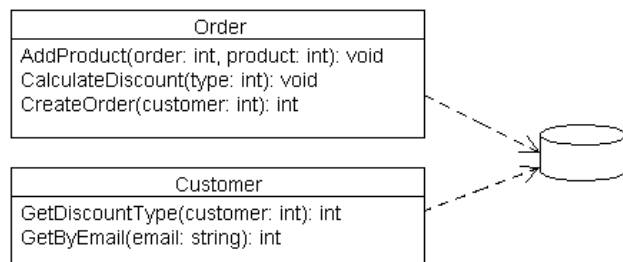


Figure 1: Class diagram for a business layer using the table module pattern

Table modules are quite simple and easy to implement. They generally work well when your domain is simple and maps directly to the tables in your database, and you're able to use ADO.NET DataSets.

Pros:

- Designed to work well with ADO.NET DataSets. DataSets use this pattern themselves, so it is quite easy to implement (eg. a DTO DataSet with no DataAdapters can be returned).
- Easier to manage complexity than Transaction Scripts
- Business logic can be shared between tables

Cons:

- Doesn't really use object-oriented principles. As there is one object that represents multiple rows, the table modules can't really collaborate with other table modules too well.

3.2.2 Transaction scripts

With the transaction script design pattern, each business function is represented in a procedural way. Each business object has a class in the data

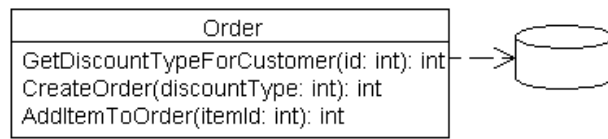


Figure 2: Class diagram for a business layer using the transaction scripts pattern

layer, and each business transaction has its own procedure in the data layer objects. This is one of the simplest ways of organising a business layer, but it often results in a lot of repetition.

Transaction scripts are good for small applications that have a small amount of business logic. As the size of the business domain grows, so too will the number of transaction scripts. These can become very hard to manage and keep consistent (code duplication will result).

Pros:

- Simple to understand – As each business transaction is a separate procedural method, it is quite easy to see what the code is doing
- Fast initial development, as the scripts are all linear and relatively self-contained

Cons:

- With a large business domain, there will be a large number of transaction scripts. These can be hard to manage.
- Hard to share business logic

3.2.3 Domain model

The domain model pattern represents each entity as a POCO (Plain Old CLR Object) class. All the database access logic is abstracted away, and a nice object-oriented view of the database is provided to the user.

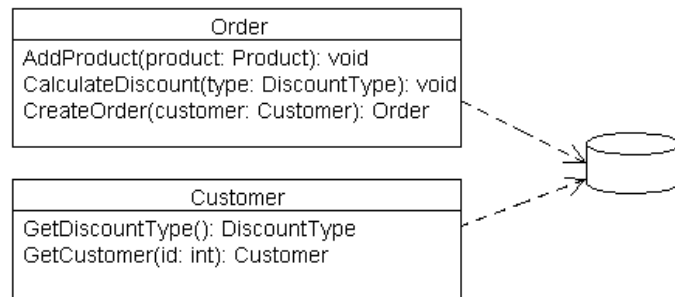


Figure 3: Class diagram of a business layer using the domain model pattern

Objects interact with each other in an object-oriented fashion – Using methods of other objects. One disadvantage of this pattern is the

mapping between the database and objects can be rather complex, although this is often solved by using a framework or toolkit like NHibernate or Entity Framework.

The domain model pattern is great for complex systems, which have a lot of business rules involving collaboration between different entities. As it uses object-oriented principles, the models can collaborate with each other if required.

Pros:

Other Portfolio Stuff

- Uses object-oriented principles. In an object-oriented language, this feels more “natural” compared to the other patterns
- Complexity can be better managed

Cons:

- Can't use ADO.NET DataSets. This is not really an issue when using an ORM library, however.

3.2.4 Service layer

A service layer is a layer that sits on top of a domain model. It abstracts away from the exact business logic implementation method, providing a “transaction script”-like view of the application. The logic is implemented in the service layer, however, exactly *how* it's implemented doesn't matter to the consumer (this could be using one of the aforementioned patterns, or a different pattern not mentioned in this portfolio).

Pros:

- Abstracts away from internal structure of domain model

Cons:

- Additional layer to maintain
- Performance may not be as good as other patterns, as another layer is being added)

3.3 User Interface Layer

The topmost layer of a three-layer architecture is a user interface layer. This is the layer that the user sees and interfaces with, and hence, contains a lot of the “shiny” features that are visible to the user. Like the other layers, the UI layer is not dependent on any implementation of the business layer. It should be able to be swapped with a totally different business layer implementation, and still work correctly. Also like the other layers, the User Interface only communicates with the layer directly below it (business logic layer); it **never** uses the data access layer directly.

There are two broad categories of user interface layers – Client-side applications, and web applications. Below these broad categories, there are more specific technologies.

3.3.1 Client-side application

Building a client-side applications is the “traditional” approach for developing software applications. This involves developing an application that the client will run on their computer. These are the perfect choice for programs that are 100% offline, or applications that need a “rich” / very responsive user interface. In the land of .NET, the most common client-side application technologies are Windows forms, and Windows Presentation Foundation.

3.3.1.1 Windows Forms

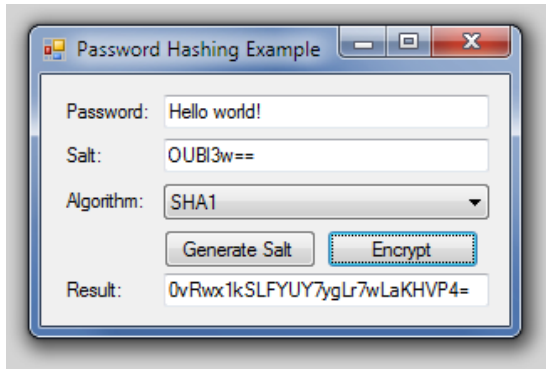


Figure 4: Example of a GUI built with Windows Forms

Windows Forms was the first GUI toolkit for the .NET Framework. It was loosely based on Microsoft's older GUI technologies, such as Forms 2.0 for Microsoft Office, and the GUI toolkit included in Visual Basic 6.0. Designing the forms is done via a designer in Visual Studio.

Unlike Visual Basic 6 (where the code for the form is all hidden), the Visual Studio GUI designer actually generates code to create the form and all its controls. This means you can look at the code

that creates the GUI, to see how it works and what it's actually doing. The code is all stored in a "partial class", meaning it doesn't get in the way when you're writing your code.

The Form class is the base of all Windows Forms applications. The Form class, as the name infers, is a form in your application. The form class then contains a number of controls (all of which have the "Control" class as a base). The controls are responsible for rendering themselves – In a way, they're self-contained. You can also make your own custom controls, which can be easily reused in multiple Windows Forms applications.

Like previous GUI technologies, Windows Forms uses an event-driven programming paradigm. Whenever something happens on the form (for example, a mouse click or drag, or a keypress), an event is fired. You can attach event listeners to these events, in order to make useful things happen when they occur. This is more efficient than old-school GUI technologies, which required polling to check whether something had happened.

3.3.1.2 WPF (Windows Presentation Foundation)

Windows Presentation Foundation is a newer GUI technology from Microsoft. Instead of using generated code, WPF uses an XML-based file format called XAML to store information about the GUI and its components. I have not really used WPF in the past so do not really know much about its advantages and disadvantages.

3.3.2 Web-based

More recently, web-based applications have started to become more popular. These are quite nice as they are not dependent on the client operating system, and can be used from remote locations (without having to install software on every computer the client uses). For a lot of database-driven applications where most of the processing is done on a server somewhere, web-based applications are a natural approach to development.

3.3.2.1 ASP.NET Web Forms

A detailed writeup on ASP.NET web forms is attached separately as a short report.

3.3.2.2 ASP.NET MVC

The MVC pattern is an alternative design pattern to the classical three-tier architecture. However, the ASP.NET MVC implementation could also be considered a “UI layer”, as it can be used on top of an existing business layer and data access layer implementation. See the short report on MVC for more details on the MVC pattern.

Following on from the popularity of MVC frameworks, both for other languages (Ruby on Rails, CodeIgniter and CakePHP for PHP, etc.) and for .NET (Castle MonoRail and Spring Framework.NET), Microsoft decided to create their own MVC framework.

ASP.NET MVC has some major advantages over ASP.NET web forms:

- It is unit testable. Controllers are easily instantiated as they do not go through the complex ASP.NET Page Life Cycle.
- It supports URL routing – Instead of ugly URLs, URLs can be short and nice.
- HTML is all hand-written, meaning it’s a lot lighter and easier to style than the often ugly autogenerated code output by ASP.NET web forms
- No ViewState fields bloating the page size!

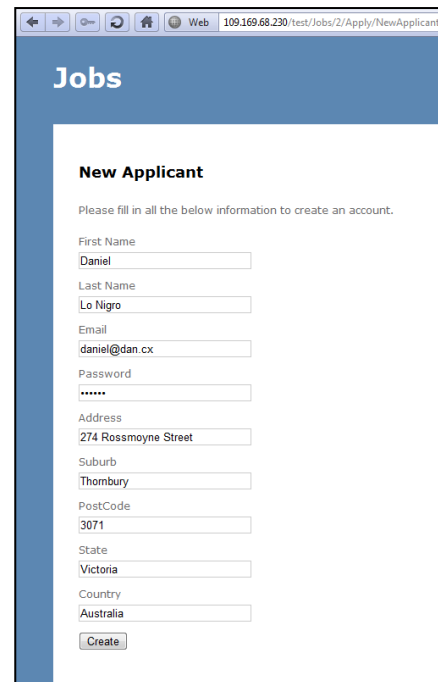
A screenshot of a web browser displaying a form titled "Jobs" with a sub-section "New Applicant". The form asks for personal information to create an account. The fields are: First Name (Daniel), Last Name (Lo Nigro), Email (daniel@dan.cx), Password (masked with dots), Address (274 Rossmoyne Street), Suburb (Thornbury), PostCode (3071), State (Victoria), and Country (Australia). A "Create" button is located at the bottom of the form.

Figure 5: Example of a web-based form, a common part of a web application

4 Web Services

With the increasing usage of the internet and web-based applications came web services. Essentially, web services are a method of communicating with remote servers using well-known standards.

As they use standard HTTP requests, web services are almost always fully interoperable between operating systems and frameworks (and custom client code is not required). A web service written in ASP.NET running on a Windows server can be consumed by a Java client running on a Linux computer with no issues. Security does not need to be message-level, it can be provided using standard HTTP over SSL (otherwise known as HTTPS), which means custom encryption is not needed (and helps with interoperability).

4.1 Protocols

A number of different web service protocols are available for use today. Some of the most common are:

4.1.1 SOAP

SOAP is an XML-based message protocol. A SOAP message contains two parts, a header and a body. The header contains metadata (such as any authentication information), and the body contains the actual payload.

4.1.1.1 WSDL

WSDL documents are used to describe SOAP web services, and their methods. WSDL documents consist of 5 sections (Wikipedia 2011b):

- Services
- Ports (Endpoints in WSDL 2.0)
 - Defines address (“connection point”) to web service
 - Generally a HTTP or HTTPS URL
- Bindings
 - Defines interface, SOAP binding style, and transport (protocol, usually HTTP)
- Messages
 - Information about messages needed to perform operations
- Types
 - XML schema to describe custom types used for the SOAP calls.
 - Can either be inline, or reference to a schema URL

4.1.1.2 Discovery

So you’ve built an awesome web service... How are people going to find it? This issue is what led to the development of UDDI (Universal Description Discovery and Integration). UDDI is essentially a public registry where you could list your web services’ WSDL documents, and other people could use the UDDI service to “discover” your web services.

UDDI was originally proposed as a web standard, however, it did not gain the traction its designers had hoped it would gain (Wikipedia 2011a). Back in 2006, Microsoft, IBM and SAP have all closed down their public UDDI servers, and the UDDI system slowly faded away.

Today, it still exists, but is very rarely used. Recently, Google have introduced their own discovery service for their APIs that is very similar to UDDI² except it doesn’t use XML or describe SOAP services.

4.1.2 REST

REST, unlike SOAP, is not a protocol as such. It is actually an architecture. REST (as it applies to HTTP) is basically a number of constraints. As long as these constraints are followed, the service can be referred to as RESTful (Richardson & Ruby 2007). The main constraints are the following:

- **HTTP Verbs** – HTTP verbs should be used for the corresponding actions. GET for retrieving data, PUT and POST for storing data, and DELETE for deleting data.
- **URIs** – URIs should be clean and point to resources. They should not have any verbs in them.
- **Stateless** – Client-server communication is stateless, and every request from the client contains all the information required to perform the request
- **Cacheable** – Clients should be able to cache the results of most GET requests. Resources should therefore define themselves as cacheable and specify an appropriate expiration time

The below table shows an example of how certain URIs would be handled based on the HTTP verb used:

² See <http://googlecode.blogspot.com/2011/05/google-apis-discovery-service-one-api.html>

URI	GET	PUT	POST	DELETE
/Products	Retrieve a list of all products, including their URLs	Replace the entire products collection with a new collection	Create a new product and return its URL	Delete all the products
/Products/123	Retrieve product 123	Replace product 123	Replace product 123	Delete product 123

The output from a REST web service can use any format, but generally either JSON or XML is used.

4.2 .NET Technologies

There are several .NET technologies that implement these various web service protocols. Some of the most common are listed below:

4.2.1 ASMX Web Services

ASMX web services (otherwise known as “ASP.NET Web Services”) are the classic method of creating SOAP web services using the ASP.NET technology. This technology was Microsoft’s initial attempt at creating a web service framework. With ASP.NET Web Services, you simply need to create a C# class with methods for each web service method, and the framework will automatically (and dynamically) generate the corresponding WSDL document at runtime. To consume the service, you simply use the “Add Web Reference” feature in Visual Studio, which will automatically generate a proxy class used to call the web service’s methods.

It is a legacy technology that’s not really recommended for new development. It’s quite an old technology and many advances have been made since then. New .NET web service development should be done with a newer technology, such as WCF.

4.2.2 Windows Communication Foundation (WCF)

WCF was originally introduced as a part of .NET Framework 3.0 (formerly known as the WinFX project). It is a newer technology, and hence is Microsoft’s currently recommended framework for web services. It supports asynchronous (also known as non-blocking) calls, and one-way calls (for example, to a message queue). WCF can be used to implement both SOAP and REST web services.

WCF has quite a number of benefits over the old ASP.NET web services technology. One major advantage is that it is transport-agnostic. While it can be used to create HTTP-based web services, WCF also supports multiple other methods of transport, including named pipes, MSMQ (see section 6) and Net.tcp (a .NET-specific, fast, compressed binary protocol)

Additionally, WCF doesn’t depend on IIS (unlike ASP.NET web services). WCF services have multiple hosting options, including inside IIS, in a windows service, in a normal application (console or even Winforms or WPF), or via Windows Process Activation³. This, combined with the protocol-independent and transport-agnostic API, makes WCF incredibly flexible. You can change the hosting method at any time, and you can change the transport you’re using simply by editing your service’s configuration file – Your code is never dependent on any particular transport being used.

³ <http://technet.microsoft.com/en-us/library/cc735229.aspx>

4.3 Security concerns

I've written about security concerns for web services in a separate short report.

5 COM+ (Assignment 4 Task 2)

COM+ is an evolution of the older COM technology from Microsoft. It allows for distributed transactions (via DCOM – Distributed COM) and resource pooling. One of the benefits of COM+ is that it allows code to run “out of process” (outside your application, in a separate process).

The component services administration UI has several options:

- **Activation** – Whether the component is activated in the caller's context or not. Depending on how the server is compiled, this may require you to recompile the server. A number of conditions need to be met in order to allow the component to be activated in the caller's context (Microsoft 2007).
- **Security** – Whether the client has to be authenticated to use the component. This can be changed without recompiling the server, but may require some changes to the client.
- **Pooling** – Pooling is when instances of a component are kept active, ready for use by any client that requests the component. This can have significant performance benefits, for example if the component takes a while to initialise. This option can be changed without recompiling or breaking the client or server.

6 MSMQ (Microsoft Message Queues)

MSMQ is a technology used for communicating between several different systems. The systems could be running on the same machine, or they could be distributed. Communication is usually one-way, with each message containing all the data required for a single business transaction.

6.1 Advantages

There are a number of advantages of using Microsoft Message Queues for communication:

- **Highly scalable** – Many machines can run in parallel, all processing different incoming messages
- **Highly available** – Messages can be posted, even when the server is offline. If the server is offline when messages are posted, they are queued for delivery as soon as it comes back online
- **Asynchronous** – As the messages are only one-way, the system doesn't need to wait for a reply.

7 Windows Services

Windows services are basically long-running tasks with no user interface. They are generally used for background processing tasks, where a user interface would not be required. Windows itself has a number of built-in services, including things like search indexing (indexing your hard drive so that searches are fast), IIS web server, and time synchronisation. Common uses for services include

things like MSMQ processing applications (see MSMQ section, above), WCF communication hosts, any other sort of background processing, and so on.

Windows services are similar to normal applications, except they have multiple entry points. Instead of a single Main method in a static class, services inherit from the **ServiceBase** class and override two methods:

- **OnStart** – This method is called when the service is started. Like the Main method of a normal Windows application, this method is passed an array of command-line arguments. Unlike a normal Windows application, the command-line arguments are very rarely useful. Generally, this method creates a new thread for the service to execute in. The OnStart method **must** return, so the service generally needs to run in a different thread.
- **OnStop** – This method is called when the service is stopped. The closest equivalent in a normal Windows Forms application would be the FormClosing event on a Form. Generally, this method would stop the thread the service is running on, and run any cleanup code, including closing any shared resources and flushing any log files.

Additionally, there are several other methods that can be overridden to provide additional functionality to the service. Notably, to enable pausing and resuming the service, the **CanPauseAndContinue** property needs to be set to **true**, and the following two methods need to be overridden:

- **OnPause** – This method is called when the service is paused. This would generally stop the thread the service is running on and save its state somewhere. While all resources used by the service don't have to be released, this method will generally release some of them (if possible)
- **OnContinue** – This method is called when the service is continued ("unpaused"). This would generally restart the service thread, and load its state that was previously saved in the OnPause method. The service should resume from the exact state it was left in when it was paused.

Note that pause and continue support is optional. Some services don't have a use for it and hence will not implement it.

7.1 Issues

7.1.1 Showing progress

One of the most common issues encountered when designing a service is how to show progress to the user. Since there is no user interface, it is quite hard to actually display data to the user. One possible solution to this is to have some sort of application that interacts with the service, and displays status information retrieved from the service.

7.1.2 Error handling

Following on from the first issue, another issue with windows services is handling of unexpected exceptions. With a normal application, unhandled exceptions will display an error message and the application will crash. With a service, this is normally not acceptable, as services will often be running on a server in the background (nobody will ever see the error message alert).

One solution to this is to have a global exception handler for the whole service, and use the **error log** to log any errors. If the service is database driven, an alternative to using the error log is to use a database for storing errors. In addition to storing the errors, for serious exceptions, quite often emails are sent to the developers informing them of the exception.

8 References

Blakeley, JA, Campbell, D, Muralidhar, S & Nori, A 2006, 'The ADO.NET entity framework: making the conceptual level real', *SIGMOD Rec.*, vol. 35, no. 4, pp. 32-39,

Fowler, M 2002, *Patterns of Enterprise Application Architecture*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Hanselman, S 2010, *Extending NerdDinner: Exploring Different Database Options*, viewed 2011-06-01, <<http://www.hanselman.com/blog/ExtendingNerdDinnerExploringDifferentDatabaseOptions.aspx>>.

Microsoft 2007, Q261096: *How to activate a COM+ component in its caller's context*, viewed 2011-06-04, <<http://support.microsoft.com/kb/261096>>.

Microsoft 2010, *Boxing and Unboxing (C# Programming Guide)*, viewed 2011-05-31, <<http://msdn.microsoft.com/en-us/library/yz2be5wk.aspx>>.

Richardson, L & Ruby, S 2007, *RESTful web services*, O'Reilly Media.

Thompson, E 2007, *Inheritance in the Entity Framework*, viewed 2011-06-04, <<http://blogs.msdn.com/b/adonet/archive/2007/03/15/inheritance-in-the-entity-framework.aspx>>.

Wikipedia 2011a, *Universal Description Discovery and Integration - Wikipedia, The Free Encyclopedia*, viewed 2011-05-17, <http://en.wikipedia.org/wiki/Universal_Description_Discovery_and_Integration>.

Wikipedia 2011b, *Web Services Description Language - Wikipedia, The Free Encyclopedia*, viewed 2011-05-15, <<http://en.wikipedia.org/wiki/WSDL>>.

Assignments

```
1 using System;
2 using System.Configuration;
3 using System.Data.Common;
4 namespace Assignment03.Task2
5 {
6     class Program
7     {
8         static void Main(string[] args)
9         {
10             ConnectionStringSettings connectionString = ConfigurationManager.ConnectionStrings[
11                 "myDatabase"];
12
13             Console.WriteLine("{0} with connection string {1}\n\n", connectionString.ProviderName,
14                 connectionString.ConnectionString);
15
16             DbProviderFactory providerFactory = DbProviderFactories.GetFactory(connectionString.
17                 ProviderName);
18             DbConnection conn = providerFactory.CreateConnection();
19             conn.ConnectionString = connectionString.ConnectionString;
20             conn.Open();
21
22             DbCommand command = conn.CreateCommand();
23             command.CommandText = @"
24                 SELECT g.Name, t.GadgetType AS GadgetType, m.Name AS ManufacturerName
25                 FROM Gadget g
26                 LEFT OUTER JOIN GadgetType t ON g.GadgetType = t.GadgetTypeID
27                 ORDER BY g.Name";
28             DbDataReader reader = command.ExecuteReader();
29             while (reader.Read())
30             {
31                 Console.WriteLine("{0} by {1} ({2})", reader["Name"], reader["ManufacturerName"], reader
32                     ["GadgetType"]);
33             }
34             reader.Dispose();
35             command.Dispose();
36             conn.Dispose();
37
38             Console.ReadKey();
39         }
40     }
41 }
42
```

```

C:\Users\Daniel\Documents\Why Dropbox\Unl\HIT3099 ... \3\Assignment03\Assignment03_Task3\Program.cs 1
1 using System;
2 using System.Collections.Generic;
3 using System.Data.Linq;
4 using System.Linq;
5 using Assignment03_Task3.QueryRecorder;
6
7 namespace Assignment03_Task3
8 {
9     class Program
10    {
11        /// <summary>
12        /// Connection string for the database
13        /// </summary>
14        protected const string CONNECTION_STRING = @"Data Source=DANIEL-LAPTOP2\SQLEXPRESS;Initial
15        Catalog=gadgets;Integrated Security=true;Pooling=false";
16        /// <summary>
17        /// How many items to display per page
18        /// </summary>
19        protected const int PER_PAGE = 5;
20
21        /// <summary>
22        /// Main entry point to the application
23        /// </summary>
24        /// <param name="args"></param>
25        static void Main(string[] args)
26        {
27            GadgetDataContext context = new GadgetDataContext(CONNECTION_STRING);
28            //context.Log = Console.Out;
29            QueryRecorder queryRecorder = new QueryRecorder.QueryRecorder();
30            context.Log = queryRecorder;
31
32            // Tell LINQ-to-SQL to join to manufacturer and type when loading, as we need data
33            // from both these tables.
34            DataLoadOptions loadOptions = new DataLoadOptions();
35            loadOptions.LoadWithGadgets(gadget => gadget.Manufacturer1);
36            loadOptions.LoadWithGadgets(gadget => gadget.GadgetType1);
37            context.LoadOptions = loadOptions;
38
39            int page = 1;
40
41            do
42            {
43                queryRecorder.Reset();
44
45                Console.WriteLine("=====");
46                int totalCount = context.Gadgets.Count();
47                int offset = PER_PAGE * (page - 1);
48                // Get a list of gadgets from the DB
49                IEnumerable<Gadget> gadgets = context.Gadgets.Skip(offset).Take(PER_PAGE);
50                foreach (Gadget gadget in gadgets)
51                {
52                    Console.WriteLine("{0} by {1} ({2})", gadget.Name, gadget.Manufacturer1.Name, gadget.
53                    GadgetType1.GadgetTypeName);
54                }
55
56                // Check if the user wants to view the queries performed
57                Console.WriteLine("Queries used: {0}. View queries? ", queryRecorder.Queries.Count);
58                char key = Console.ReadKey().KeyChar;
59                Console.WriteLine();
60                if (key == 'y')
61                {
62                    OutputQueries(queryRecorder);
63                }
64
65                // Prompt user for new page
66                Console.WriteLine("Page {0} of {1}. Go to page: ", page, Math.Ceiling(totalCount / (PER_PAGE
67                * 1.0)));
68                string input = Console.ReadLine();
69                if (!int.TryParse(input, out page))
70                    break;
71            } while (true);
72
73            context.Dispose();
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97

```

```

C:\Users\Daniel\Documents\Why Dropbox\Unl\HIT3099 ... \3\Assignment03\Assignment03_Task3\Program.cs 2
72    }
73    }
74    /// <summary>
75    /// Output a list of queries to the console
76    /// </summary>
77    /// <param name="queryRecorder"><see cref="QueryRecorder" /> used to record the queries</param>
78    static void OutputQueries(QueryRecorder queryRecorder)
79    {
80        foreach (QueryInfo info in queryRecorder.Queries)
81        {
82            Console.WriteLine(info.Query);
83            if (info.Params.Count > 0)
84            {
85                Console.WriteLine("Parameters:");
86                foreach (KeyValuePair<string, object> kvp in info.Params)
87                {
88                    Console.WriteLine("{0} = {1}", kvp.Key, kvp.Value);
89                }
90            }
91            Console.WriteLine("-----");
92        }
93    }
94    }
95    }
96    }
97

```

```

C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099 ... \Assignment03.Task3\QueryRecorder\QueryRecorder.cs 1
1 using System.Collections.Generic;
2 using System.IO;
3 using System.Text;
4 using System.Text.RegularExpressions;
5
6 namespace Assignment03.Task3.QueryRecorder
7 {
8     /// <summary>
9     /// Record all the queries performed by a LINQ to SQL data context
10    /// </summary>
11    /// <author email="daniel@dan.cx">Daniel Lo Nigro</author>
12    public class QueryRecorder : TextWriter
13    {
14        protected static Encoding _encoding;
15        /// <summary>
16        /// Gets the list of queries ran so far.
17        /// </summary>
18        public IList<QueryInfo> Queries { get; private set; }
19        /// <summary>
20        /// Information about the latest query executed
21        /// </summary>
22        protected QueryInfo _latestQuery;
23        /// <summary>
24        /// Regular expression for matching parameters in the output
25        /// </summary>
26        protected static Regex _paramRegex = new Regex(@"(-- (\S+):(.+) \[(.+)\]", RegexOptions.Compiled);
27
28        /// <summary>
29        /// Initializes a new instance of the <see cref="QueryRecorder"> class.
30        /// </summary>
31        public QueryRecorder()
32        {
33            Reset();
34        }
35
36        /// <summary>
37        /// Reset the query listing to a blank list
38        /// </summary>
39        public void Reset()
40        {
41            Queries = new List<QueryInfo>();
42        }
43
44        /// <summary>
45        /// Get the encoding from this TextWriter. Required in any TextWriter implementation
46        /// </summary>
47        public override Encoding Encoding
48        {
49            get { return _encoding ?? (Encoding = new UnicodeEncoding()); }
50        }
51
52        /// <summary>
53        /// Write a line to the stream. Grab the query information from the line written, and pull
54        /// it apart, retrieving the query and parameters it had
55        /// </summary>
56        /// <param name="value"></param>
57        public override void WriteLine(string value)
58        {
59            // NOTE: This is EXTREMELY implementation-specific
60            // Ignore short lines (like line breaks)
61            if (value.Length <= 6) return;
62
63            string firstFour = value.Substring(0, 4);
64            // Skip the random info at the bottom of the output
65            if (firstFour == "--C")
66                return;
67
68            // Is it a parameter?
69            if (firstFour == "--@"")
70            {
71                Match match = _paramRegex.Match(value);
72                // Parameters always come after queries so we can associate this
73                // parameter with the latest query
74                _latestQuery.Params.Add(match.Groups[1].Value, match.Groups[3].Value);

```

```

C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099 ... \Assignment03.Task3\QueryRecorder\QueryRecorder.cs 2
75     }
76     }
77     {
78         _latestQuery = new QueryInfo { query = value };
79         Queries.Add(_latestQuery);
80     }
81     }
82 }
83 }
84

```


C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099 ... \Assignment03.Task3\QueryRecorder\QueryInfo.cs 1

```
1 using System.Collections.Generic;
2
3 namespace Assignment03.Task3.QueryRecorder
4 {
5     public class QueryInfo
6     {
7         public string Query { get; set; }
8         public IDictionary<string, object> Params { get; set; }
9
10        public QueryInfo()
11        {
12            Params = new Dictionary<string, object>();
13        }
14    }
15 }
16
```

C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099 ... \Assignments\4\Assignment04_Task1_Client\Program.cs 1

```
1 using System;
2 using System.Messaging;
3
4 namespace Assignment04_Task1_Client
5 {
6     class Program
7     {
8         protected const string MESSAGEQUEUE_NAME = @".\private\Assignment04";
9         static void Main(string[] args)
10        {
11            // Create the queue if it doesn't already exist
12            if (MessageQueue.Exists(MESSAGEQUEUE_NAME))
13                MessageQueue.Create(MESSAGEQUEUE_NAME);
14
15            MessageQueue queue = new MessageQueue(MESSAGEQUEUE_NAME);
16
17            Console.WriteLine("Enter messages, or a blank line to quit.");
18
19            while (true)
20            {
21                string text = Console.ReadLine();
22
23                // Quit the loop if the user entered nothing
24                if (text == string.Empty)
25                    break;
26
27                // Create a new message with the specified text
28                queue.Send(text);
29            }
30        }
31    }
32 }
33
```

C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099 ...Assignments\A\Assignment04_Task1.Server\Service1.cs 1

```
1 using System;
2 using System.IO;
3 using System.Messaging;
4 using System.ServiceProcess;
5 using System.Threading;
6
7 namespace Assignment04_Task1.Server
8 {
9     public partial class Service1 : ServiceBase
10    {
11        protected const string MESSAGEQUEUE_NAME = @".\Private$\Assignment04";
12        protected const string LOG_FILE = @"C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099 Enterprise .NET\Assignments\A\Log.txt";
13        protected Thread _thread;
14
15        public Service1()
16        {
17            InitializeComponent();
18        }
19
20        protected override void OnStart(string[] args)
21        {
22            _thread = new Thread(Run);
23            _thread.Start();
24        }
25
26        protected override void OnStop()
27        {
28            // Aborts are bad, but this will do for now.
29            _thread.Abort();
30        }
31
32        protected void Run()
33        {
34            MessageQueue queue = new MessageQueue(MESSAGEQUEUE_NAME);
35            // Open the log file with shared read/write so other processes can still read it.
36            FileStream logFile = new FileStream(LOG_FILE, FileMode.Append, FileAccess.Write, FileShare.ReadWrite);
37            StreamWriter logWriter = new StreamWriter(logFile);
38
39            while (true)
40            {
41                Message message = queue.Receive();
42                message.Formatter = new XmlMessageFormatter(new[] {typeof(string)});
43                logWriter.WriteLine("{1}: Received '{0}'", message.Body, DateTime.Now);
44                logWriter.Flush();
45            }
46        }
47    }
48 }
49
```

C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099 ... \Assignments\4\Assignment04_Task2\TestComClass.cs 1

```
1 using System.EnterpriseServices;
2 using System.Messaging;
3
4 namespace Assignment04_Task2
5 {
6     public class TestComClass : ServiceComponent
7     {
8         protected const string MESSAGEQUEUE_NAME = @".\private\Assignment04";
9
10        public string Test()
11        {
12            return "Hello world!!";
13        }
14
15        public void PostMessage(string message)
16        {
17            MessageQueue queue = new MessageQueue(MESSAGEQUEUE_NAME);
18            queue.Send("Via COM: " + message);
19        }
20    }
21 }
22
```

C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099NET\Assignments\4\Assignment04_Task2\test.vbs 1

1 Set oTest = CreateObject("Assignment04_Task2_TestComClass")

2 MsgBox oTest.Test()

3 oTest.PostMessage("Hello world from VBScript")

4

C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099 ... Assignments\5\Assignments05.Task1\HelloWorld.asmx.cs 1

```
1 using System.Web.Services;
2
3 namespace Assignment05.Task1
4 {
5     [WebService(Namespace = "http://dan.cx/Uni/Enterprise.NET/Assignment05/Task1")]
6     [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
7     [System.ComponentModel.ToolboxItem(false)]
8     // To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following
9     // line.
10    [System.Web.Script.Services.ScriptService]
11    public class HelloWorld : WebService
12    {
13        [WebMethod]
14        {
15            public string Hello()
16            {
17                return "Hello World";
18            }
19        }
20    }
21    [WebMethod]
22    {
23        public int Blah()
24        {
25            return 42;
26        }
27    }
28 }
```

C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099NET\Assignments\5\Assignment05_Task2\Program.cs 1

```
1 using System;
2 using System.Xml.Linq;
3 using Assignment05_Task2.AustralianPostCode;
4
5 namespace Assignment05_Task2
6 {
7     class Program
8     {
9         static void Main(string[] args)
10        {
11            AustralianPostCodeSoapClient client = new AustralianPostCodeSoapClient();
12            string result = client.GetAustralianPostCodeByLocation("Thornbury");
13            //Console.WriteLine(result);
14
15            /* This Australian Postcodes service is very weird - It returns XML We need to parse
16             * the XML in order to actually use it. It'd be better to return normal objects instead
17             */
18            XElement xml = XElement.Parse(result);
19            foreach (XElement row in xml.Elements("Table"))
20            {
21                Console.WriteLine("{0} - {1}", row.Element("PostCode").Value, row.Element("Location").
22                Value);
23            }
24        }
25    }
26 }
27 }
28
```

C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099 ... \Assignments\5\Assignment05_Task3_Client\Program.cs 1

```
1 using System;
2 using Assignment05_Task3_Client.HelloWorldService;
3 namespace Assignment05_Task3_Client
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             HelloWorldServiceClient client = new HelloWorldServiceClient();
10            var result = client.HelloWorld();
11            Console.WriteLine(result.Message);
12            Console.ReadKey();
13        }
14    }
15 }
16 }
17 }
```


C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099 ... \5\Assignment05.Task3.Service\HelloWorldService.cs 1

```
1 namespace Assignment05.Task3.Service
2 {
3     /// <summary>
4     /// Actual implementation of the Hello World service
5     /// </summary>
6     public class HelloWorldService : HelloWorldService
7     {
8         public HelloWorldType HelloWorld()
9         {
10            return new HelloWorldType { Message = "Hello world!" };
11        }
12    }
13 }
14
```

C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099 ... \5\Assignment05.Task3.Service\HelloWorldService.cs 1

```
1 using System.Runtime.Serialization;
2 using System.ServiceModel;
3
4 namespace Assignment05.Task3.Service
5 {
6     /// <summary>
7     /// Service contract for the Hello World service
8     /// </summary>
9     [ServiceContract]
10    public interface IHelloWorldService
11    {
12        [OperationContract]
13        HelloWorldType HelloWorld();
14    }
15
16    /// <summary>
17    /// Simple data contract
18    /// </summary>
19    [DataContract]
20    public class HelloWorldType
21    {
22        [DataMember]
23        public string Message { get; set; }
24    }
25 }
26
```

```
1 using System;
2 using Assignment05.Task3.Service;
3 namespace Assignment05.Task3.ServiceHost
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             System.ServiceModel.ServiceHost host = new System.ServiceModel.ServiceHost(typeof
10                (HelloWorldService));
11             host.Open();
12             Console.WriteLine("Service running!");
13         }
14     }
15     Console.ReadLine();
16     host.Close();
17 }
18 }
19 }
20 }
```

```
1 using System;
2 using System.Windows.Forms;
3 namespace Assignment07_Task1
4 {
5     public partial class FrmMain : Form
6     {
7         protected PasswordHasher _hasher = new PasswordHasher();
8
9         public FrmMain()
10        {
11            InitializeComponent();
12        }
13    }
14
15    private void FrmMain_Load(object sender, EventArgs e)
16    {
17        algorithm.SelectedIndex = 1;
18    }
19
20    private void generateSalt_Click(object sender, EventArgs e)
21    {
22        salt.Text = _hasher.GenerateSalt();
23    }
24
25    private void encrypt_Click(object sender, EventArgs e)
26    {
27        result.Text = _hasher.ComputeHash(password.Text, salt.Text);
28    }
29
30    private void algorithm_SelectedIndexChanged(object sender, EventArgs e)
31    {
32        _hasher.HashAlgorithm = _hasher.HashAlgorithmFactory((string)algorithm.SelectedItem);
33    }
34 }
35 }
36
```

```

1 using System;
2 using System.Security.Cryptography;
3 using System.Text;
4 namespace Assignment07_Task1
5 {
6     public class PasswordHasher
7     {
8         /// <summary>
9         /// How many bytes to use in generated salts
10        /// </summary>
11        protected int SALT_BYTE_LENGTH = 4;
12
13        /// <summary>
14        /// Gets or sets the hash algorithm.
15        /// </summary>
16        /// <value>
17        /// The hash algorithm.
18        /// </value>
19        public HashAlgorithm HashAlgorithm { get; set; }
20
21        /// <summary>
22        /// Generates a random salt value
23        /// </summary>
24        /// <returns>Random salt value</returns>
25        public string GenerateSalt()
26        {
27            byte[] saltBytes = new byte[SALT_BYTE_LENGTH];
28            // Use secure random number generator
29            RNGCryptographyServiceProvider random = new RNGCryptographyServiceProvider();
30            random.GetNonZeroBytes(saltBytes);
31            return Convert.ToBase64String(saltBytes);
32        }
33
34        /// <summary>
35        /// Computes the hash for a password, using a particular salt
36        /// </summary>
37        /// <param name="password">The password.</param>
38        /// <param name="salt">The salt.</param>
39        /// <returns>Hash of the password</returns>
40        public string ComputeHash(string password, string salt)
41        {
42            string saltedPassword = AddSaltToPassword(password, salt);
43            byte[] inputBytes = Encoding.ASCII.GetBytes(saltedPassword);
44            byte[] hash = HashAlgorithm.ComputeHash(inputBytes);
45            return Convert.ToBase64String(hash);
46        }
47
48        /// <summary>
49        /// Add the salt to the password. Separate method so implementation can be easily changed.
50        /// Virtual so it can be overridden by child classes.
51        /// </summary>
52        /// <param name="password">The password</param>
53        /// <param name="salt">The salt</param>
54        /// <returns>Salted password</returns>
55        protected virtual string AddSaltToPassword(string password, string salt)
56        {
57            return password + salt;
58        }
59
60        public HashAlgorithm HashAlgorithmFactory(string algorithmName)
61        {
62            //return (HashAlgorithm) Activator.CreateInstance(typeof(System.Security.Cryptography.
63            + algorithmName + "Managed"));
64            switch (algorithmName.ToUpper())
65            {
66                case "SHA1": return new SHA1Managed();
67                case "SHA256": return new SHA256Managed();
68                case "SHA384": return new SHA384Managed();
69                case "SHA512": return new SHA512Managed();
70                case "MD5": return new MD5CryptographyServiceProvider();
71                default:
72                    throw new ArgumentException("Invalid hash algorithm specified", "algorithmName");
73            }
74        }
75    }
76 }
77

```

```

74        }
75    }
76 }
77

```

```
1 using System;
2 using System.Security;
3 using System.Security.Permissions;
4 using System.Security.Principal;
5
6 namespace Assignment07.Task2
7 {
8     Class Program
9     {
10         static void Main(string[] args)
11         {
12             AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);
13             try
14             {
15                 AwesomeFunction();
16             }
17             catch (SecurityException ex)
18             {
19                 Console.WriteLine("Sorry, you're not awesome: {0}", ex.Message);
20             }
21
22             Console.ReadKey();
23         }
24
25         /// <summary>
26         /// ONLY awesome users can execute this method (need to be in "AwesomeUsers" windows group)
27         /// </summary>
28         [PrincipalPermission(SecurityAction.Demand, Role="AwesomeUsers")]
29         static void AwesomeFunction()
30         {
31             Console.WriteLine("You have permission to be awesome.");
32         }
33     }
34 }
35
```

Spikes

Enterprise .NET

Spike Outcome Report

By Lakthinda Ranasinghe / Daniel Lo Niegro

Transaction Scripts

The name "Transaction Scripts" was new to us. But after researching about it, We realised that it is not entirely an alien concept, and it seems that we have already used that concept in our regular developments without knowing, that we are using a well recognised design pattern.

So, what is "Transaction Scripts"? It is a design pattern used to implement a business logic as a procedure, with one or more transactions with a database and, transactions involve with steps of validation and calculation of data. Transaction Scripts pattern can be modularised, and used in any layer, depend on the complexity of the business logic. It works as an independent procedure, hence, changes to the logic would not affect the other business logics in a program. It is recognised as best suitable for implementing simple logic and small applications.

Implementation

Business Logic - Meeting Room Booking System

We initiated the project by identifying different business logics where we can apply Transaction Scripts in the design. We decided to feature following features of the booking system.

1. Meeting room booking

Related business rules:

- a. A room can only be booked by 1 person at a time.
- b. A meeting record must have a booking owner, a reason for the meeting, time/date and duration.
- c. When a meeting is booked, it should check for conflicts and notify user.

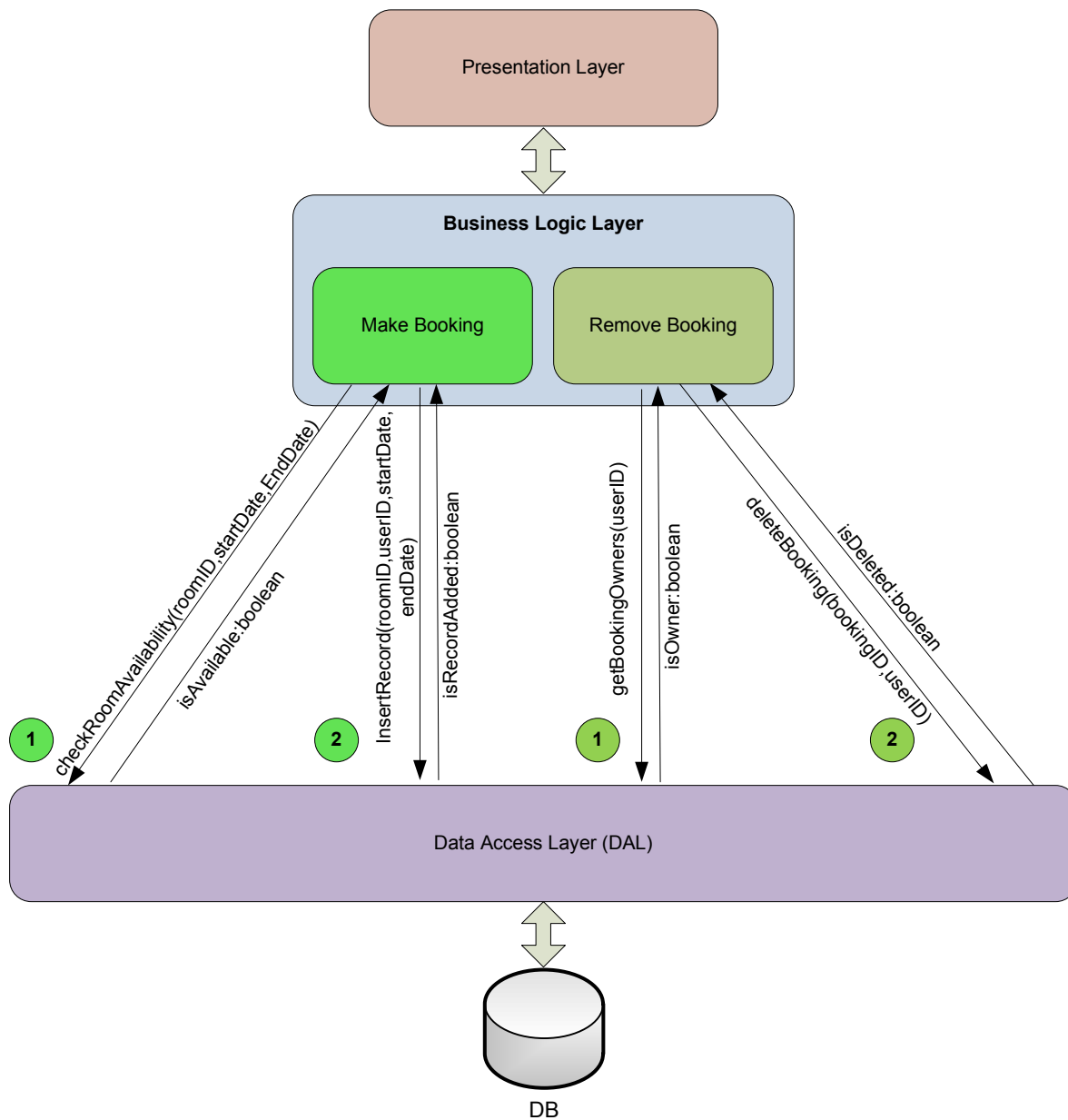
In Meeting room booking, we first check existing booking records for any conflicts, and if there is any, the user will be notified and would not make the booking. If there are no conflicts, create a booking record in "meeting room table".

2. Remove a booking

Related business rules:

- a. A meeting can only be cancelled, by the person who booked it.

In Remove a booking, first check whether the user is the owner of the booking, and if not notify the user. secondly, if the user is the owner of the booking, then remove the record from meeting room table.



Development

Step 1. Create the database. Either import an SQL DDL file, or follow following steps to create a new database:

1. Open Visual Studio
2. Right-click “Data Connections”
3. Click “Create New SQL Server Database”
4. Enter .SQLEXPRESS as the server name, and choose a database name. Click OK
5. Create database according to required structure.

Step 2. Create four projects in Visual Studio. These are:

- DataAccess
- BusinessLayer
- Common
- UserInterface

Step 3: Create the DataSet in Common project

Common will contain things that are common to multiple layers.

1. Right-click “Common” project and select Add -> New Item
2. Select “DataSet”, type an appropriate name and click OK. The DataSet designer should open
3. In Data Connections pane on the left of Visual Studio, expand your database -> Tables
4. Drag all the tables onto the DataSet designer
5. Add appropriate TableAdapter methods by right-clicking the TableAdapters and clicking “Add Query” This includes things like simple SELECT statements (eg. GetRoomByID - SELECT * FROM Room WHERE ID = @id) as well as more advanced SELECT statements using joins (eg. GetMeetingsByRoomIdAndDate:

```
SELECT m.*
FROM Meeting m
INNER JOIN Room r ON m.RoomID = r.id
WHERE r.ID = @roomId AND m.StartDate <= @startDate OR m.EndDate >= @endDate
```

Step 4: Create data access layer. Create classes for each database entity (corresponding to the entities in the DataSet) for CRUD operations, as well as any other TableAdapter methods you’ve created. Create interfaces **first** (in the Common project), and then create classes that implement these interfaces (in the DataAccess project)

Step 5: Create business logic layer.

This is where we use the transaction scripts. As explained above, we will implement the above defined transactions as follows:

a) Make Booking

```
public bool MakeBooking(int roomId, int userId, DateTime startDate, DateTime endDate, string
reason)
{
    // Business rule: Clashes (multiple meetings at same time) are not allowed
    if (CheckRoomAvailability(roomId, startDate, endDate))
        throw new Exception("Room is already booked at this time!");
    // Business rule: Add booking record with userID,startDate,endDate,reason
    return _meetingData.Insert(roomId, userId, startDate, endDate, reason);
}

public bool CheckRoomAvailability(int roomId, DateTime startDate, DateTime endDate)
{
    // Check if any meetings already exist
    RoomBooking.MeetingDataTable dataTable =
    _meetingData.GetMeetingsByRoomIdAndDates(roomId, startDate, endDate);

    return (dataTable.Count > 0);
}
```

b) Delete Booking

```
public void CancelMeeting(int meetingId, int userId)
{
    RoomBooking.MeetingRow meeting = _meetingData.GetMeeting(meetingId);

    // Business rule: Meetings can only be cancelled by the user that created them
    if (meeting.OwnerUserID != userId)
        throw new Exception("Meetings can only be cancelled by the user that created them");

    // Business rule: Delete the booking
    _meetingData.Delete(meetingId);
}
```

Step 6: Set up Castle Windsor in the UI project. This involves adding a reference to Castle Windsor, and creating the App.config file. Refer to Dependency Injection / Inversion of Control spike for more details.

Summary

We implemented a Meeting room booking system using Transaction Scripts with two functions, Adding a booking and a deleting a booking. We use the modularised approach to implement above program, where we use Interfaces to access the business logic layer and the data access layer, allowing us to have flexibility over business logic implementation without affecting other layers.

Outstanding Issues

- What is the best way to apply Transaction Script?

Technology Requirements

- Visual Studio 2010
- .NET Framework 3.5

Links/Resources

- Google - <http://google.com/>
- Notes from Database Programming lectures (Swinburne University)
- Patterns of Enterprise Application Architecture (Martin Fowler), ISBN 0321127420
- <http://kostas.homeip.net/reading/Software%20Development/Design%20Patterns/Patterns%20of%20Enterprise%20Application%20Architecture/transactionScript.html>

```
1 using System;
2 using Spike8.BusinessLogic.Abstract;
3 using Spike8.Common;
4 using Spike8.DataAccess.Abstract;
5
6 namespace Spike8.BusinessLogic
7 {
8     public class MeetingBL : IMeetingBL
9     {
10         protected readonly IRoomData _roomData;
11         protected readonly IMeetingData _meetingData;
12
13         public MeetingBL(IRoomData roomData, IMeetingData meetingData)
14         {
15             _roomData = roomData;
16             _meetingData = meetingData;
17         }
18
19         public bool MakeBooking(int roomId, int userId, DateTime startDate, DateTime endDate, string
20             reason)
21         {
22             // Business rule: Clashes (multiple meetings at same time) are not allowed
23             if (!CheckRoomAvailability(roomId, startDate, endDate))
24                 throw new Exception("Room is already booked at this time!");
25
26             return _meetingData.Insert(roomId, userId, startDate, endDate, reason);
27         }
28
29         public bool CheckRoomAvailability(int roomId, DateTime startDate, DateTime endDate)
30         {
31             // Check if any meetings already exist
32             RoomBooking.MeetingDataTable datatable = _meetingData.GetMeetingsByRoomIdAndDates(roomId,
33                 startDate, endDate);
34             return datatable.Count != 0;
35         }
36
37         public void CancelMeeting(int meetingId, int userId)
38         {
39             RoomBooking.MeetingRow meeting = _meetingData.GetMeeting(meetingId);
40             // Business rule: Meetings can only be cancelled by the user that created them
41             if (meeting.OwnerUserId != userId)
42                 throw new Exception("Meetings can only be cancelled by the user that created them");
43             _meetingData.Delete(meetingId);
44         }
45     }
46 }
```



Faculty of Information and Communication Technologies

Enterprise .NET

Spike Outcome Report

Willson Santoso

Daniel Lo Nigro

Darren Pratt

Distributed Transactions with Windows Communication Foundation Services

1. BACKGROUND

What are transactions?

A transaction is any modification to any record in a database table. When you insert, update, or delete a record, it is called a transaction. On the other hand, when you select a record to read, it is not a transaction, because nothing changes in the database.

Some business rules require that a group of modifications be treated as a single transaction. For example, in accounting software, a debit and a credit must succeed or fail together because the accounts must "balance." If a debit succeeds and the credit fails, the accounts will not balance. This kind of balanced modification collectively represents a single transaction. If an error occurs in the transaction, the entire transaction must be rolled back.

A transaction is initiated from a single program. This type of program executes a command to begin and commit (complete) a transaction. Between these two commands, the program can modify any

quantity of records within the database. This kind of multiple-step modification represents a single transaction.

What are Distributed Transactions?

A distributed transaction involves more than one database. A single distributed transaction may involve a company's inventory, customer relations management, and accounts receivable databases. For example, when an order is received on a Web site, the program may need to modify product inventory levels in one database, customer profile information in another database, and an account balance in yet another database.

Databases are generally hosted on more than one networked computer, so a distributed transaction is not only across application, but also across platforms. In Enterprise systems, we may not be communicating directly with a database, but with services, which in turn connect to a database. Thus services fall into the scope of distributed transactions. Many aspects of a distributed transaction are identical to a transaction whose scope is a single database. For example, a distributed transaction provides predictable behavior by enforcing the ACID properties that define all transactions.

Transaction Scope

The TransactionScope class provides a simple way to mark a block of code as participating in a transaction, without requiring you to interact with the transaction itself. A transaction scope can select and manage the transaction automatically. You do not need to enlist resources explicitly with the transaction. Any System.Transactions resource manager can detect the existence of a transaction created by the scope and automatically enlist, as supported by the .Net framework.

Creating a transaction scope

The transaction scope is started once you create a new TransactionScope object. As illustrated in the code sample for this spike, Microsoft recommends that you create scope with a using statement, which works like a try...finally block to ensure that the scope is disposed of properly.

```
try
{
    using (TransactionScope ts = new TransactionScope())
    {
        //method to updatedb

        //perform update

        //perform other update

        //call a service

        ts.Complete();
    }
}
catch
```

```
{  
    .....
```

When you instantiate `TransactionScope`, the transaction manager determines which transaction to participate in. Once determined, the scope always participates in that transaction.

Completing a transaction scope

When your application completes all the work it wants to perform in a transaction, you should call the `Complete` method only once to inform the transaction manager that it is acceptable to commit the transaction. Microsoft recommends putting the call to `Complete` as the last statement in the using block. Failing to call this method aborts the transaction, because the transaction manager interprets this as a system failure, or equivalent to an exception thrown within the scope of transaction. However, calling this method does not guarantee that the transaction will be committed. It is merely a way of informing the transaction manager of your status.

If the `TransactionScope` object created the transaction initially, the actual work of committing the transaction by the transaction manager occurs after the last line of code in the using block. If it did not create the transaction, the commit occurs whenever `Commit` is called by the owner of the `CommittableTransaction` object. At that point the Transaction Manager calls the resource managers and informs them to either commit or rollback, based on whether the `Complete` method was called on the `TransactionScope` object.

The using statement ensures that the `Dispose` method of the `TransactionScope` object is called even if an exception occurs. The `Dispose` method marks the end of the transaction scope. Exceptions that occur after calling this method may not affect the transaction. This method also restores the ambient transaction to its previous state.

A `TransactionAbortedException` is thrown if the scope creates the transaction, and the transaction is aborted. A `TransactionIndoubtException` is thrown if the transaction manager cannot reach a `Commit` decision. No exception is thrown if the transaction is committed.

Rolling back a transaction

If you want to rollback a transaction, you should not call the **Complete** method within the transaction scope. For example, you can throw an exception within the scope. The transaction in which it participates in will be rolled back.

Creating a Distributed Transaction

You can administer distributed transactions through the Microsoft Distributed Transaction Coordinator (DTC), which is included with the Component Services administrative tool. The DTC provides services designed to ensure successful and complete transactions, even with system failures, process failures, and communication failures. Each computer participating in a distributed transaction manages its own resources and data and also acts in concert with other computers in the transaction. Above all, a distributed transaction must commit or abort its work entirely on all participating computers. The DTC performs the transaction coordination role for the components

involved and acts as a transaction manager for each computer that manages transactions.

The DTC uses the two-phase commit protocol. Phase one involves the transaction manager requesting each component to prepare to commit; in phase two, if all components successfully prepare, the transaction manager broadcasts the commit decision.

In general, transactions involve the following steps:

1. Applications call the transaction manager to begin a transaction.
2. When the application has prepared its changes, it asks the transaction manager to commit the transaction. The transaction manager keeps a sequential transaction log so that its commit or abort decisions will be durable.
 - If all components are prepared, the transaction manager commits the transaction and the log is cleared.
 - If any component cannot prepare, the transaction manager broadcasts an abort decision to all elements involved in the transaction.
 - While a component remains prepared but not committed or aborted, it is in doubt about whether the transaction committed or aborted. If a component or transaction manager fails, it reconciles in-doubt transactions when it reconnects.

DTC Components

In Microsoft Windows, the DTC is a system service that is tightly integrated with COM+. To help make distributed transactions work more seamlessly, COM+ directs the DTC on behalf of an application. This makes it possible to scale transactions from one to many computers without adding special code.

When a COM+ application requires a transaction to span multiple computers, the DTC makes it possible by generating a transaction ID, managing transaction-related communications, and enlisting resources. When the DTC enlists a resource in a transaction, it extends the transaction's protection to that resource.

To perform specialized tasks within a distributed transaction, the DTC architecture includes the following elements:

- The DTC transaction manager
- The DTC log
- The DTC proxy
- The Component Services administrative tool
- The DTC Files

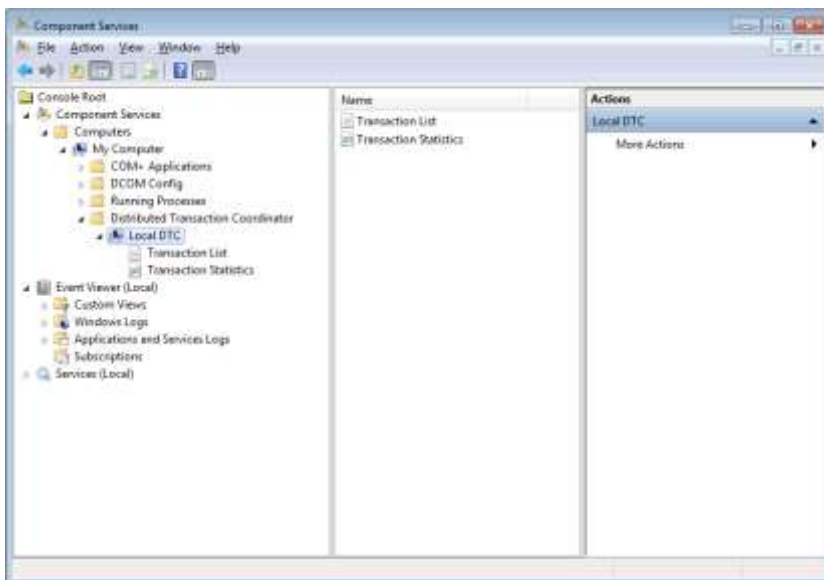
(see the reference website for further information on this)

2. CONFIGURATION

Configuring DTC and WS-Atomic Transaction for WCF distributed transactions

WCF transactions rely on Microsoft DTC (Distributed Transaction Coordinator) to coordinate the transaction between multiple services and DTC itself relies on WS-AT to communicate between the DTC instances on different machines.

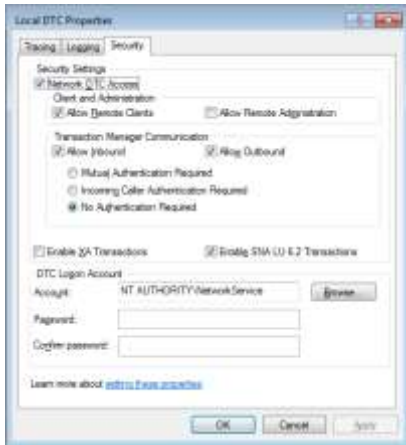
MS DTC is configured via the Component Services Console, which can be started using the `dcomcfg.exe` command from the command prompt. Once you are inside Component Service Console, browse to the **Console Root\Component Services\Computers\My Computer\Distributed Transaction Coordinator** on the tree view menu on the left and right click on the **Local DTC** and go into its property.



Go to the Security tab on the property dialog.

To ensure that DTC works over the network on multiple machines, tick the **Network DTC Access** option, tick the **Allow Remote Clients** option, and tick the **Allow Inbound** and **Allow Outbound** options.

Since this demonstration is run on a workgroup rather than on two machines within the same Active Directory domain, we will have to tick the **No Authentication Required** option as well.



Once the DTC is configured as per the above, the next step is to configure the WS-Atomic Transaction feature. We have to enable the WS-AT component by running this command line.

```
regasm.exe /codebase WsatUI.dll
```

The regasm.exe can be found in %PROGRAMFILES%\Microsoft SDKs\Windows\v6.0\Bin, and it must be run under administrator privilege.

The WS-AT component requires the use of certificates to establish trusts between machines. This involves a certificate (cert A) being issued and registered on computer A and exported to a file and imported to computer B. While on the other end, a certificate (cert B) needs to be issued and registered on computer B, and then exported to a file and imported to computer A.

To do this you must first create a root certificate which will allow you to create and sign other certificates. The command to create a root certificate looks like this.

```
makecert.exe -pe -n CN=MSDTC-Wsat-CA -cy authority -r -sv Msdtc.pvk Msdtc.cer
```

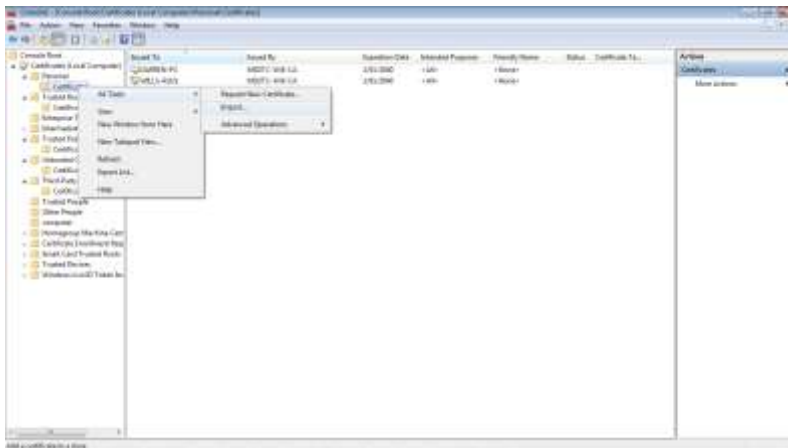
After that, install the root certificate on your machine with the following command.

```
makecert.exe -ss Root -sr LocalMachine -n CN=MSDTC-Wsat-CA -cy authority -r -sv Msdtc.pvk
```

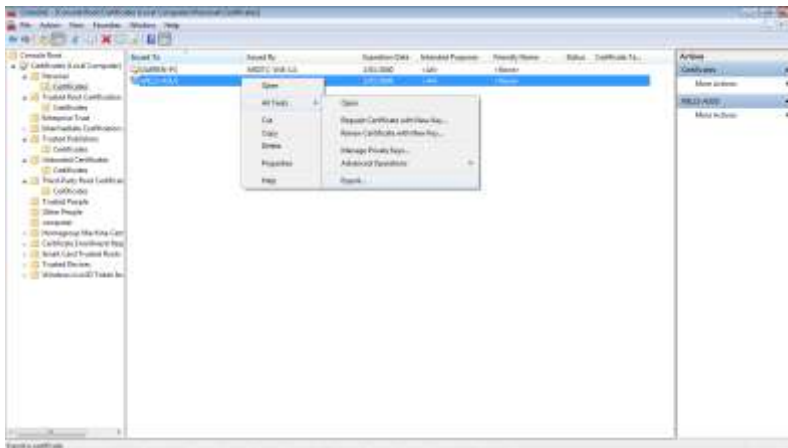
Once the root certificate is installed, you can use that root certificate to create and sign another certificate that allows different computers to exchange data. This needs to be done on both machines in our demonstration.

```
makecert -ss My -sr LocalMachine -n CN=%COMPUTERNAME%.%USERDNSDOMAIN% -sky exchange -ir LocalMachine -iv Msdtc.pvk -ic Msdtc.cer
```

Once the certificate is created, add it to your list of personal certificates. This can be done via the management console (mmc). Expand the **Certificates (Local Computer)\Personal** and the right click on **Certificate** node and click on **All Tasks\Import** on the context menu. Browse to the location where you have saved your personal certificate file created in the previous step and import it.



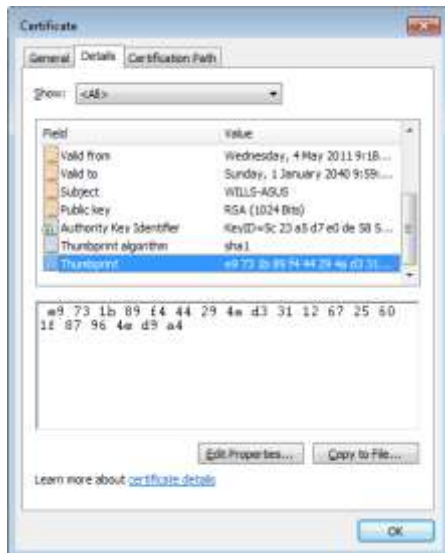
A new certificate (eg WILLS-ASUS) will be added to your personal certificate list. Export the certificate by right clicking on it and then clicking on the **All Tasks\Export** menu.



The exported cer file will need to be copied to the remote machine and then imported from that end to establish the trust. To import the cer file, simply start the MMC and expand the **Certificates (Local Computer)\Personal ** tree and then right click on the **Certificates** node and select the **All Tasks\Import** menu and browse to the cer file exported above.

Repeat the step of creating, registering and exporting a certificate on the remote computer and then import the cer file on your local computer to complete the steps of establishing the trust between the two machines.

Once the certificates have been registered on both ends, you can use wscntcfg to configure WS-AT to use these certificates. You will need to obtain the thumbprints of both certificates by double clicking on each certificate in the MMC GUI.



Copy the thumbprint into the wsatconfig command without the spaces. Use your local machine certificate thumbprint on endpointCert and the remote machine certificate on the accountCerts

```
WsatConfig.exe --network:enable --port:8443 --endpointCert:<thumbprint> -  
accountsCerts:<thumbprint"> -restart
```

The above steps complete the DTC and WS-AT configuration for this demonstration. Remember to run all the commands under admin privilege.

3. DEVELOPMENT

Building the Spike Solution

Step 1: Create the database.

Either import an SQL DDL file, or follow following steps to create a new database:

1. Open Visual Studio
2. Right-click “Data Connections”
3. Click “Create New SQL Server Database”
4. Enter .\SQLEXPRESS as the server name, and choose a database name. Click OK
5. Create database according to required structure.

Step 2: Create two WCF services

Please refer to the Windows Communication Foundation spike for more detailed information. For the endpoint, you need to use either net.tcp, or wsHttp – Other bindings do not support transactions.

If using wsHttp on a custom port, you need to allow your user to bind to the port. Run the following command to do so:

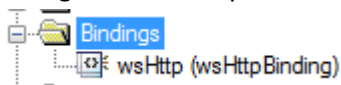
```
netsh http add urlacl url=http://+:54321/ user=COMPUTERNAME\Username
```

(where 54321 is the port number, COMPUTERNAME is the name of your computer, and Username is your username on your computer)

There are several steps required to enable transactions in a WCF service.

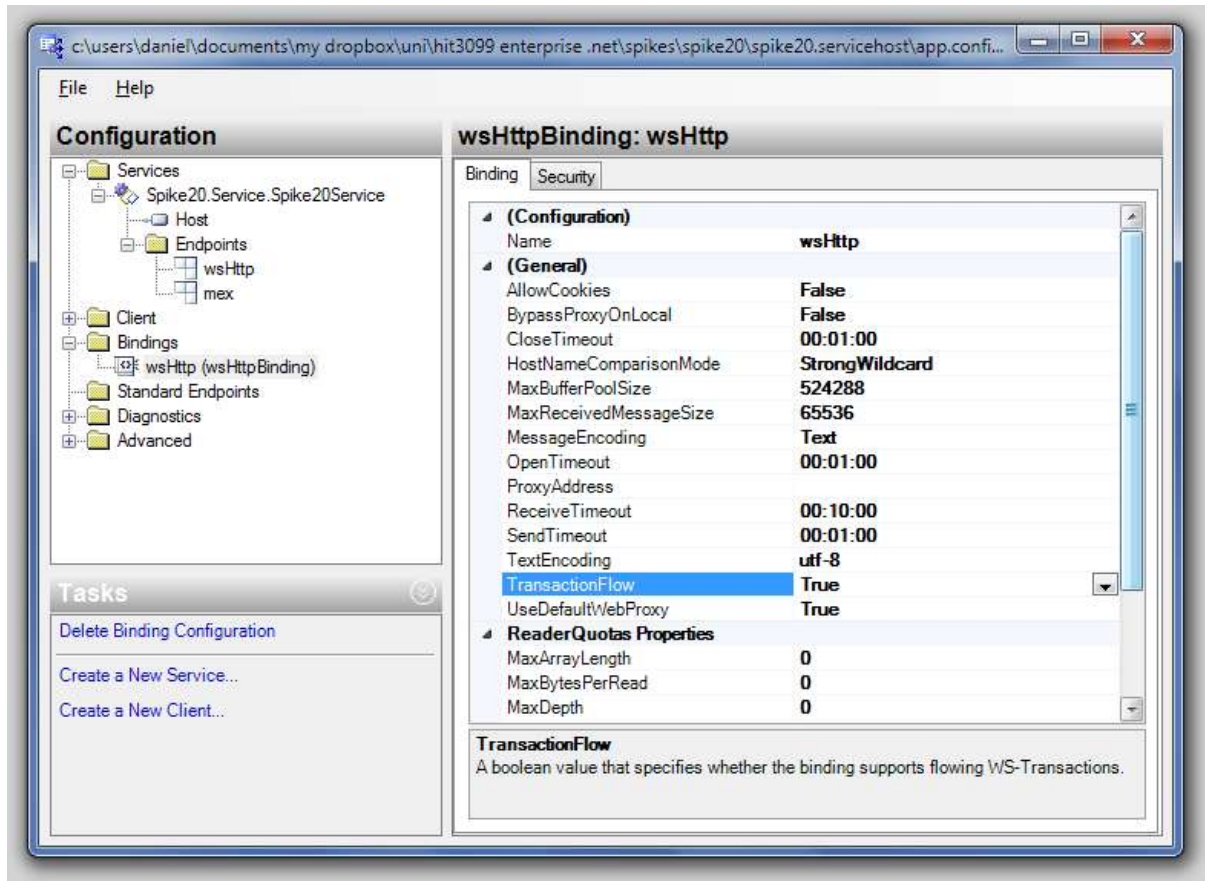
Step 3: Enable transactions in the service – Set up binding

Open the WCF config editor by right-clicking the App.config file and clicking “Edit WCF Configuration”. Expand the Bindings folder and ensure that a binding exists under it:



If there is no binding listed, right-click *Bindings*, click *New*, and add a new wsHttpBinding.

Click on the binding, and change the **TransactionFlow** property to **true**.



Click on the endpoint (*wsHttp* in the above screenshot) and set the binding to your custom binding (*wsHttp*).

Step 4: Enable transactions in the service – Add transaction support to service interface

In the service interface, add the `TransactionFlow(TransactionFlowOption.Allowed)` attribute to all methods that need to support transactions. For example:

```
[ServiceContract]
public interface ISpike20Service
{
    [OperationContract]
    [TransactionFlow(TransactionFlowOption.Allowed)]
    void InsertMessage(string message);
}
```

This tells the WCF framework that transactions are allowed to “flow” from the client into the service.

In the service implementation, add the `OperationBehavior(TransactionScopeRequired = true)` attribute to all methods that need to support transactions. For example:

```
public class Spike20Service : ISpike20Service
{
    [OperationBehavior(TransactionScopeRequired = true)]
    public void InsertMessage(string message)
    {
        // ... stuff here
    }
}
```

This tells the WCF framework that a transaction scope is required to call the method.

Step 5: Add transaction support to the client code

In your calling code, wrap all code in a `TransactionScope`. This runs all the calls in a single transaction. If the transaction is successful, call the `Complete` method on the transaction scope, which will commit all the changes performed in the transaction.

```
using (TransactionScope transaction = new TransactionScope(TransactionScopeOption.RequiresNew))
{
    try
    {
        // ... Call client methods here
        transaction.Complete();
    }
    catch (Exception ex)
    {
        // Something bad happened! Log the error.
        // If the WCF channel is faulted, we need to "reset" it
        // http://stackoverflow.com/questions/2008382/how-to-heal-faulted-wcf-channels
        if (client.InnerChannel.State == CommunicationState.Faulted)
        {
            client.Abort();
            ((IDisposable)client).Dispose();
            client = new Spike20ServiceClient();
        }

        transaction.Dispose();
    }
}
```

The last section in the “catch” is required because when a fault occurs, the WCF channel is no longer usable. In order to send more requests to the same client, a new client instance needs to be created.

References

Transaction Defined - msdn.microsoft.com/en-us/library/ms973833.aspx

Transaction Scope - msdn.microsoft.com/en-us/library/ms172152.aspx

Distributed Transactions

- msdn.microsoft.com/en-us/library/ms681291%28VS.85%29.aspx
- [msdn.microsoft.com/en-us/library/ms683623\(v=VS.85\).aspx](https://msdn.microsoft.com/en-us/library/ms683623(v=VS.85).aspx)
- support.microsoft.com/kb/316247

Transaction Promotion - msdn.microsoft.com/en-us/library/ms131083.aspx

Faulted WCF Channel

- stackoverflow.com/questions/2008382/how-to-heal-faulted-wcf-channels

WSAtomicTransactions – msdn.microsoft.com/en-us/library/ms733943.aspx

WCF – Spike 07 WCF

Makecert - www.digitallycreated.net


```
1 using System;
2 using System.ServiceModel;
3 using System.Transactions;
4 using Spike20.Client.InternalService;
5 using Spike20.Client.ServiceReferences;
6
7 namespace Spike20.Client
8 {
9     class Program
10    {
11        static void Main(string[] args)
12        {
13            Spike20ServiceClient client = new Spike20ServiceClient();
14            InternalServiceClient internalClient = new InternalServiceClient();
15
16            internalClient.ResetCount();
17
18            while (true)
19            {
20                console.WriteLine("Enter a message: ");
21                string message = Console.ReadLine();
22                using (TransactionScope transaction = new TransactionScope(TransactionScopeOption.
                RequiresNew))
23                {
24                    try
25                    {
26                        client.InsertMessage(message);
27                        internalClient.IncrementCount();
28                        transaction.Complete();
29                    }
30                    catch (Exception ex)
31                    {
32                        Console.ForegroundColor = ConsoleColor.Red;
33                        Console.WriteLine("Couldn't add that message - {0}!",
34                            ex.InnerException != null ? ex.InnerException.Message : ex.
35                            Message);
36                        Console.ForegroundColor = ConsoleColor.Gray;
37
38                        // If the WCF channel is faulted, we need to "reset" it
39                        // http://stackoverflow.com/questions/2008382/how-to-heal-faulted-wcf-channels
40                        if (client.InnerChannel.State == CommunicationState.Faulted)
41                        {
42                            client.Abort();
43                            ((IDisposable)client).Dispose();
44                            client = new Spike20ServiceClient();
45                        }
46                        transaction.Dispose();
47                    }
48                }
49                Console.WriteLine("Messages so far = {0}", internalClient.GetCount());
50            }
51        }
52    }
53 }
54
```

C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099 ...\Spikes\Spike20\Spike20.Service\ISpike20Service.cs 1

```
1 using System.ServiceModel;
2
3 namespace Spike20.Service
4 {
5     [ServiceContract]
6     public interface ISpike20Service
7     {
8         [OperationContract]
9         [TransactionFlow(TransactionFlowOption.Allowed)]
10        void InsertMessage(string message);
11    }
12 }
13
```

```
1 using System;
2 using System.ServiceModel;
3
4 namespace Spike20.Service
5 {
6     public class Spike20Service : ISpike20Service
7     {
8         [OperationContract(TransactionScopeRequired = true)]
9         public void InsertMessage(string message)
10        {
11            if (message == null)
12                throw new ArgumentException("message");
13
14            Console.WriteLine("Received InsertMessage call - {0}", message);
15
16            Spike20Entities entities = new Spike20Entities();
17            // Create both new message entities
18            Message dbMessage = new Message { MessageText = message, Date = DateTime.Now };
19            Message2 dbMessage2 = new Message2 { MessageText = message, Date = DateTime.Now };
20
21            try
22            {
23                // Try to add both message entities
24                entities.Messages.AddObject(dbMessage);
25                entities.SaveChanges();
26                entities.Messages2.AddObject(dbMessage2);
27                entities.SaveChanges();
28            }
29            catch (Exception ex)
30            {
31                Console.ForegroundColor = ConsoleColor.Red;
32                Console.WriteLine("ERROR: {0}", ex.InnerException != null ? ex.InnerException.Message :
33                    ex.Message);
34                Console.ForegroundColor = ConsoleColor.Gray;
35                // Throw the exception back to the caller
36                throw;
37            }
38        }
39    }
40 }
41
```

C:\Users\Daniel\Documents\My Dropbox\Uni\HIT3099NET\Spike\Spike20\Spike20.ServiceHost\Program.cs 1

```
1 using System;
2 using Spike20.Service;
3
4 namespace Spike20.ServiceHost
5 {
6     class Program
7     {
8         static void Main(string[] args)
9         {
10            System.ServiceModel.ServiceHost host = new System.ServiceModel.ServiceHost(typeof
                (Spike20.Service));
11            host.Open();
12            Console.WriteLine("Service running");
13            Console.ReadLine();
14            host.Close();
15        }
16    }
17 }
18
```