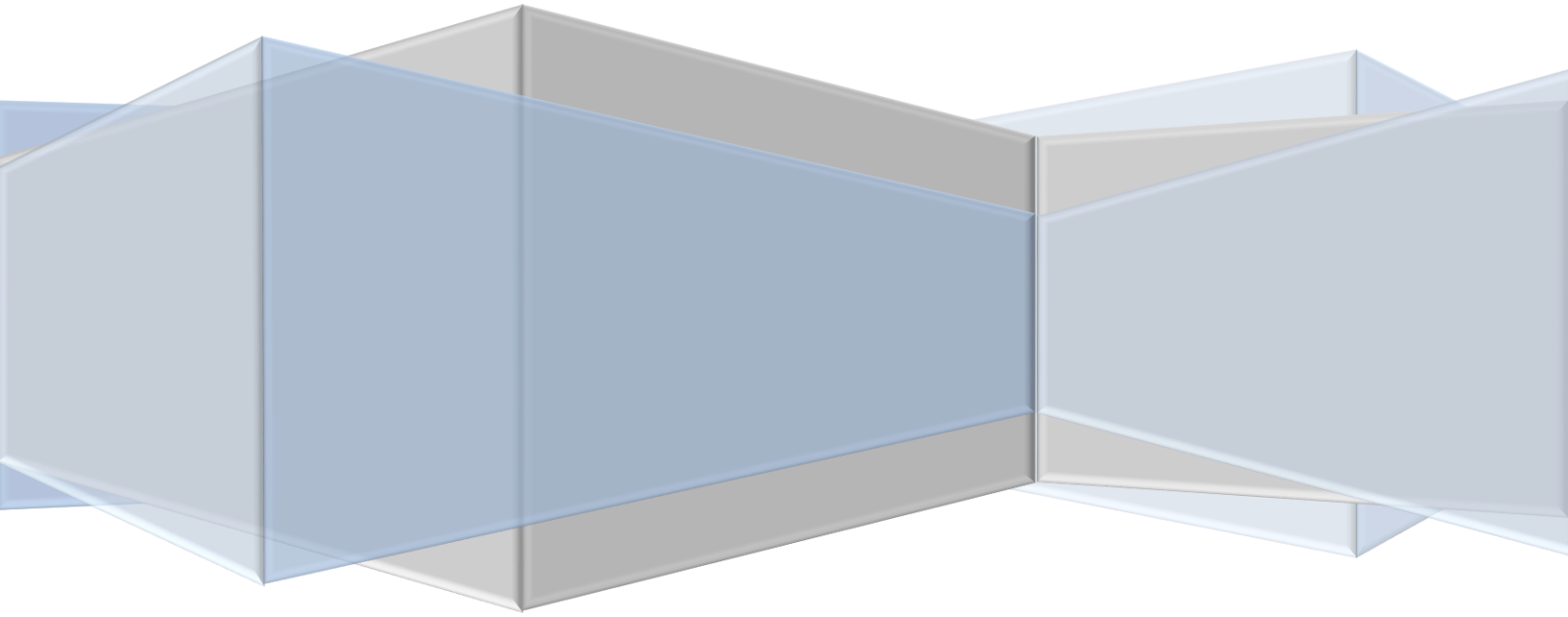
NUS Information Technology

NUS Student .NET Development & Deployment Guide

Version 1.0

Last Updated: 05 Aug 2019



Document Change History

Version	Date Changed	Summary of Changes	Change Author
1.0	05-Aug-2019	First release of the Student .NET Development and Deployment Guide	Andy Gue

Contents

1 Introduction 4
1.1 Glossary 4
1.1.1 Tools 4
1.1.2 Libraries 4
2 Use of 3rd Party Tools and Libraries 4
3 Development Environment 4
3.1 Components and Tools for Applications Development 4
4 Version Control System 5
4.1 Guidelines 5
5 Application Architecture - A Recommendation 6
5.1 Benefits of an Application Architecture 6
a) Flexible Architecture 6
b) Ease of Maintenance 6
c) Performance and Scalability 6
d) Lower Development Costs and Risks 6
5.2 ASP.NET Web Forms and ASP.NET MVC 7
5.2.1 ASP.NET MVC 7
5.2.2 ASP.NET Web Forms 7
5.2.3 Comparison between ASP.NET MVC and ASP.NET Web Forms 7
5.3 3rd Party Components 7
5.4 Control Tables 8
6 Design & Coding Guidelines 8
6.1 Design Patterns 8
6.2 Best Practices 8
6.3 Naming and Coding Convention 8
6.4 Data Access Coding Guidelines 9
6.5 Data Confidentiality 9
6.6 Data Integrity 9
6.7 Access Control 9
6.8 Audit Trail 9
6.9 Guidelines on Calling Web Services from .NET 10

6.10 Guidelines on File Upload 11
7 Testing 11
7.1 System Integration Test 11
7.2 User Acceptance Test 11

1 Introduction

This document introduces the NUS .NET Development and Deployment Guidelines and related resources. The purpose of this document is to:

a) Enable developers to jump start and get quickly assimilated to the development and deployment environment in NUS.
b) Enable developers to understand and perform the applicable processes required in developing .NET applications
c) Provide a set of guidelines for developing .NET applications
d) Define the processes to enable a stable and more manageable .NET infrastructure for NUS Information Technology (NUS IT) applications.
e) Be used as a handy reference for staff who are handling .NET application development.

1.1 Glossary

1.1.1 Tools

Tools refer to software or utilities (but not limited) that you use to assist in your development process. Examples of IDE tools are Microsoft Visual Studio, etc.

1.1.2 Libraries

Libraries refer to software libraries or external DLLs, written by someone else and provided in an API form for developers to consume and it accomplish certain functions. Examples of libraries are iTextSharp, etc.

2 Use of 3rd Party Tools and Libraries

Application teams or developers are encouraged to use or utilize the list of recommended tools or libraries.

If there is a better tool or library that is not mentioned in the recommended list, and with good reason to use it, please justify your decision to your team.

3 Development Environment

3.1 Components and Tools for Applications Development

The following summarizes the components and tools required in projects development.

#	Components and Tools	Details

1	IDE (Integrated Development Environment)	For new .NET development, it is recommended to use the latest version of Microsoft Visual Studio. • Visual Studio Code (https://code.visualstudio.com/) For existing projects, it is recommended to use the version that is being used.
2	Database Driver	Depending on the database that is being used, developers have to install the relevant driver.
3	Versioning Control	Developers can use Git or Apache Subversion (SVN) to manage their source code versions. It's highly recommended to use GitHub Desktop to simplify your version control management process.

4 Version Control System

The source code should be managed by a version control system. The recommended version control is Git, and we highly recommend using GitHub Desktop for a more streamlined version control management process.

4.1 Guidelines

The followings are some guidelines when using Git.

a. A typical application has two main types of branches: **master** and **supporting branches.**
The **master** stores stable releases of the application. Our source code at **origin/master** should always be at production-ready state; whereas **the supporting branches** are used to store unfinished versions whose enhancements or bug-fixes require relatively longer time to complete.
Read more at https://nvie.com/posts/a-successful-git-branching-model/. *Not sure if we are allowed to use links to third-party websites?*

b. Git is primarily meant for storing application's source codes and other deployment-requisite files (HTML, JSP, JavaScript, XML, API jar files, etc); developers should abstain from using it to store huge binary files, or as storage for office documents such as Excel, Word, etc. Also practice using .gitignore if there are any files in your source code that are large, but are expected to remain unchanged throughout your development process, for example: node_modules in your npm package. Read more at https://github.com/github/gitignore/tree/master/Global.

The followings are some guidelines when using SVN.

c. A typical application has 3 SVN folders, namely **tag**, **branch** and **trunk**.
The **tag** stores stable releases of the application; whereas **branch** is used to store unfinished versions whose enhancements or bug-fixes require relatively longer time to complete.

Lastly, the **trunk** stores current production copy of application and is used for auto-deploying the application to various operating environments.

d. The SVN is primarily meant for storing application's source codes and other deployment-requisite files (HTML, JSP, JavaScript, XML, API jar files, etc); developers should abstain from using it to store huge binary files, or as storage for office documents such as Excel, Word, etc.

5 Application Architecture - A Recommendation

In broad term and with respect to application design and development context, Application Architecture is about structures and patterns. It defines the key components that are typically required to support enterprise application. For further reading on Application Architecture, you may check out:
http://msdn.microsoft.com/en-us/library/dd673617.aspx
http://www.derekashmore.com/2012/01/benefits-of-standardized-application.html

5.1 Benefits of an Application Architecture

a) Flexible Architecture

By having a well-layered and component design, it allows the architecture to be evolved over time. Having loose coupling and separation of concerns within an architecture, the components can be swap in and out with other technologies without affecting the entire architecture.

b) Ease of Maintenance

With a good component design, it is also easier to maintain the application. Complexity is largely hidden by abstraction and encapsulation. Complicated logic are written once but used many times. Isolation of bugs is made easier because culprit components can be more easily identified.

c) Performance and Scalability

By adopting proven frameworks, the application architecture can be assured to be designed with performance and scalability in mind. These frameworks, implemented with best practice design patterns, form a strong foundation for an enterprise application implementation.

d) Lower Development Costs and Risks

With the application architecture, development costs and risks will also be significantly lower. Through the use of encapsulation, developers will focus on implementing business functionality while a team of senior developers will develop the architecture framework to support the

application. Reusability is high and bad programming practices are more or less confined to the business layer.

5.2 ASP.NET Web Forms and ASP.NET MVC

Depending on the complexity and scale of the application and developer's competency, ASP.Net application can be developed using MVC or Web Forms. Each methodology has its advantages and disadvantages.

Below is a list of references and comparison between ASP.Net Web Forms and MVC:

5.2.1 *ASP.NET MVC*

NUS IT recommends ASP.NET MVC to be used in applications. MVC structures the application into three main components: the model (M), the view (V) and the controller (C). The framework is ideal for developers who want full control over the behavior of an application.
Useful Reference: http://www.asp.net/mvc

5.2.2 *ASP.NET Web Forms*

The Web Forms-based framework uses Page Controller pattern that adds functionality to individual pages. It provides a large number of components for rapid application development (RAD).

Useful Reference: http://www.asp.net/web-forms

5.2.3 *Comparison between ASP.NET MVC and ASP.NET Web Forms*

Useful References:
http://msdn.microsoft.com/en-us/magazine/dd942833.aspx
http://www.codeproject.com/Articles/528117/WebForms-vs-MVC
http://www.codeproject.com/Articles/38778/ASP-NET-WebForms-and-ASP-NET-MVC-in-Harmony

5.3 3rd Party Components

This section highlights 3rd party libraries that developers have feedback that are popular, good or useful. We list them here for developers to consider using, to solve the problems they encounter.

No.	Recommend Library	Usage
1.	iTextSharp	For PDF generation and manipulation.
3.	Log4net	For purposes of application debugging and auditing.

5.4 Control Tables

Hard coding should be avoided at the design phase. Access control lists and application parameters should be placed into database control tables. If possible, it is recommended to move all the configuration keys from web.config to control table in order to get identical web.config in the test and production servers.

6 Design & Coding Guidelines

Having consistent application design and coding conventions facilitates the ease of maintenance and readability of applications codes. It also encourages more structural discipline in the coding.

The following sections highlight some of the .NET Design Patterns, Best Practices and Coding Guidelines.

6.1 Design Patterns

Refer to http://www.dofactory.com/Patterns/Patterns.aspx

6.2 Best Practices

Refer to https://docs.microsoft.com/en-us/aspnet/aspnet/overview/web-development-best-practices/what-not-to-do-in-aspnet-and-what-to-do-instead

6.3 Naming and Coding Convention

Refer to https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines

a. There are two appropriate ways to capitalize identifiers, depending on the use of the identifier: PascalCasing and camelCasing.
b. The PascalCasing is used for all identifiers except parameter names. It capitalizes the first character of each word (including acronyms over two letters in length). For example, PropertyDescriptor, HtmlTag and IOStream
c. The camelCasing is used for parameter names only. It capitalizes the first character of each word except the first word. For example, propertyDescriptor, isStream and iStream.
d. Do favour readability over brevity. For example, use CanScrollHorizontally rather than ScrollableX
e. Do not use underscores, hyphens or any other non-alphanumberic characters.
f. Do not use any acronyms that are not widely accepted. However, acronyms like NUS is acceptable in NUS context.
g. Names of Namespaces:
Nus.(<Faculty>|<Department>).(<Product>|<Technology>)[.<SubNamespace>]. For example Nus.RegistrarOffice.Cors

6.4 Data Access Coding Guidelines

The following DOs and DON'Ts on data access and coding are recommended:

a. Avoid PL/SQL. Put the logics and computation process in the application codes.
b. Do not use cookies to store data. Encrypt cookies if you have to use it.
c. SQL should be optimized; get expert advices on complex SQL queries. Developers or application teams should monitor and review with DBA for in-efficient SQLs.
d. Fields used for search criteria should be index especially when querying large chunk of data.
e. Never embed database access statements in static pages and JSPs. Data access has to be encapsulated in the backend layers with data access objects accessing the databases.
f. All open connections database must be closed in the finally block of a try-catch statement to prevent connection leak.
g. Implement audit logging for important/critical transactions and activities.
h. Formulate/Keep database connection string in registry.

6.5 Data Confidentiality

Confidentiality is "the ability to ensure that an asset is *viewed* only by authorised parties." No unauthorised disclosure of information.
No passwords or sensitive data are to be hardcoded into the source codes. Data access methods have to be encapsulated within data access objects or through the Spring framework's best practices.

6.6 Data Integrity

Integrity is "the ability to ensure that an asset is *modified* only by authorised parties." No unauthorised modification of information or processes.
It is mandatory for applications to perform server-side validation of data before processing the data for computation or storage. Though client side data validation may be in place but it can be bypassed by turning off script features in the browser. Hence, data types and data integrity should be check before submitting to the server, for example input values for NRIC, age, gender can be validated. Input values in application forms open up the risk for Cross Site Scripting (XSS), as such, it should also be validated using the Apache Commons or the ESAPI API.

6.7 Access Control

Applications should have in place some access control (e.g. role-based access control) to manage the intended users and their actions on the application data. Users are commonly demarcated into either admin group or normal user group in accordance to their rights and security roles in the application. For example, use least privilege principle: "only access rights that are *required* to complete the role will be assigned." No access right is given for tasks not required by a certain user/group.

6.8 Audit Trail

It is mandatory for applications to log user access (login/logout) to aid accountability. Application owners and developers must exercise intuition on what addition data is to be captured to provide a complete picture of user actions for accountability in the event that security audit and data-mining is required.

- Login including any failed attempts
- Privileges changes
- Roles changes
- User account changes
- Execution of any Data Definition Language including Create, Drop, Alter Table, database links, directors, indices, stored procedures, profiles, roles, etc.
- Commands that require administrator access

6.9 Guidelines on Calling Web Services from .NET

There are two types of web service available in NUS now. One is SOAP and another is non-SOAP Web API, either RESTful or not.

To call SOAP web service from Visual Studio, use “Add Service Reference” to automatically generate the proxy assembly and other reference files.

Most other non-SOAP web APIs in NUS are not RESTful, especially the old ones. It is advised to use HttpClient instead of HttpWebRequest or WebClient to call this type of web service. HttpClient is built into Visual Studio 2012 and later version so it can be referred to after you “Add Reference” to System.Net.Http.

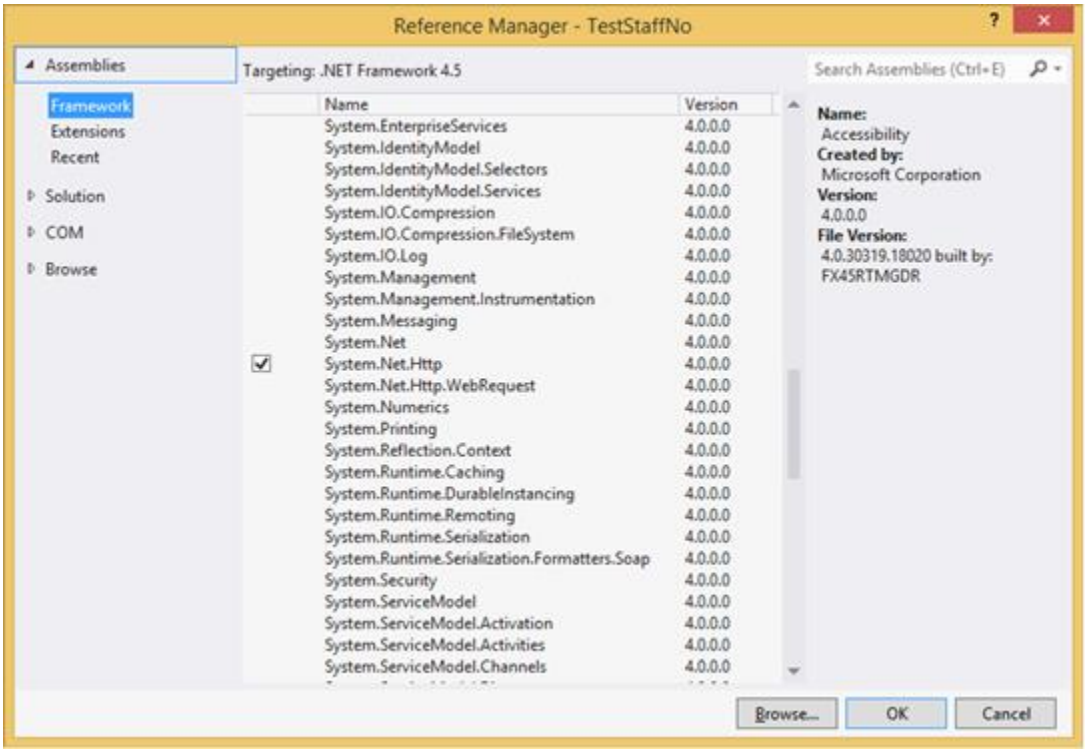


Figure 1 - Add Reference to System.Net.Http

6.10 Guidelines on File Upload

The following DOs and DON'Ts on file upload are recommended:

a. Whitelist the extensions of the files which can be uploaded.
b. Whitelist MIME type of the files by using System.Web.MimeMapping.GetMimeMapping (Visual Studio 2012 onwards).
c. Use GUID as the file name while storing the files.
d. Do not store files uploaded by the users into the application directory.
e. Application shall not keep uploaded files that are no longer in use. Such files must be removed.

7 Testing

7.1 Unit testing

Test small bits of your programme. Fast & isolated which makes it simpler to spot bugs in your code.

7.2 Regression testing

When you just add in a new feature, re-test to check that the new codes have not affected your existing working features.

7.3 System Integration Test

System Testing evaluates the behavior of a complete and fully integrated software product based on the software requirements specification (SRS). It is black box type of testing which does not require knowledge of internal design, structure or code and totally based on the user's point of view.

Processes:

- Create System Integration test plan.
- Create test cases.
- Create test data used for System Integration Test.
- Run the test cases.
- Bug reporting, bug verification & regression testing.

7.4 User Acceptance Test

User Acceptance Test is based on user requirements specification and should be written by the user. It specifies all functionalities expected of the system are tested and working as expected. Processes:

- Analyse business requirements.
- Create UAT test plan.
- Create UAT test cases.
- Prepare test data (production like data).
- Run the test cases.
- Record the results.
- Confirm business objectives.

~ End of document ~