NUS Information Technology

NUS Student Java Development & Deployment Guide

Version 1.1

Last Updated: 10-Dec-2019

Document Change History

Version	Date Changed	Summary of Changes	Change Author
1.0	05-Aug-2019	First release of the Student Java Development and Deployment Guide	Andy Gue
1.1	10-Dec-2019	Make slight modifications to the Student Java Development and Deployment Guide	Wong Shen Kai Dinh Nho Bao

Contents

1 Introduction

This document introduces the NUS Java Development and Deployment Guidelines (JDDG) 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 Java applications
c) Provide a set of guidelines for developing Java applications
d) Define the processes to enable a stable and more manageable Java infrastructure for NUS Information Technology (NUS IT) applications.
e) Be used as a handy reference for staff who are handling Java application development.

1.1 Glossary

1.1.1 Tools

Tools refer to software, utilities (and other programs) that you use to assist in your development process. Examples of IDE tools are IntelliJ IDEA, Eclipse, Visual Studio Code, NetBeans, etc.

1.1.2 Libraries and frameworks

Libraries refer to Java software libraries and frameworks usually packaged as .zip or .jar files, written by someone else and provided in an API form for developers to consume. Examples of Java libraries and frameworks are Spring, Hibernate, Apache Log4J, Quartz, Gson, 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 project development.

#	Components and Tools	Details

1	Application Server	To prevent compatibility issues during deployment, developers should install the same version of Application Server that is being used in the Production environment.
2	Java SE Development Kit	To prevent compatibility issues during deployment, developers should install the same version of JDK that is being used in the Production environment. Developers must also ensure their deployed applications can run properly in the Production environment and refrain from customizing the applications' builds using specific versions of JDK or database driver, other than those installed in the application server. Download from https://www.oracle.com/technetwork/java/javase/downloads/index.html
3	Database Driver	Depending on the database that is being used, developers have to install the relevant driver.
4	IDE (Integrated Development Environment)	For Java development, it is recommended to use the latest stable builds of IntelliJ IDEA, Eclipse, Visual Studio Code or NetBeans. • IntelliJ IDEA (https://www.jetbrains.com/idea/) • Eclipse (http://www.eclipse.org/) • Visual Studio Code (https://code.visualstudio.com/) • NetBeans (https://netbeans.org/)
5	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.

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 MVC Architecture

MVC, or model view controller, is an acronym for a high-level architectural pattern that covers the separation of user interface (view), data and business services (model) and the interactions between them (controller). The main principle of this pattern is to simplify the implementation of applications that need to act on user requests and manipulate and display data. There are three distinct components within the MVC pattern:

- *Model*. The business tier's domain model and its related services. This contains the data needed to render the view, which is populated by the controller. Examples are Plain Old Java Objects (POJOs), Enterprise JavaBeans (EJBs), etc.

- *View*. Provides the user interface for displaying and manipulating the model. Views should only present the application data encapsulated within the model, without including any business logic or connections to the database layer. Examples are JavaServer Pages (JSPs), Velocity, etc.

- *Controller*. The logic that orchestrates the interactions between the model and the view. It is responsible for building an appropriate model and passing it to the view for rendering. Examples are Java servlets, etc.

Figure 4 illustrates the MVC pattern and a typical view request is handled as follows:

1. *Request*: A request is submitted to the server. On the server side, most frameworks (for example, Spring, Struts, etc) will have a dispatcher (in the form of a servlet) to handle the request.

2. *Invokes*: The dispatcher dispatches the request to the appropriate controller based on the HTTP request information and the web application configuration.

3. *Service Call*: The controller interacts with the service layer.

4. *Response*: The controller updates the model and based on the execution result, returns the corresponding view to the user.

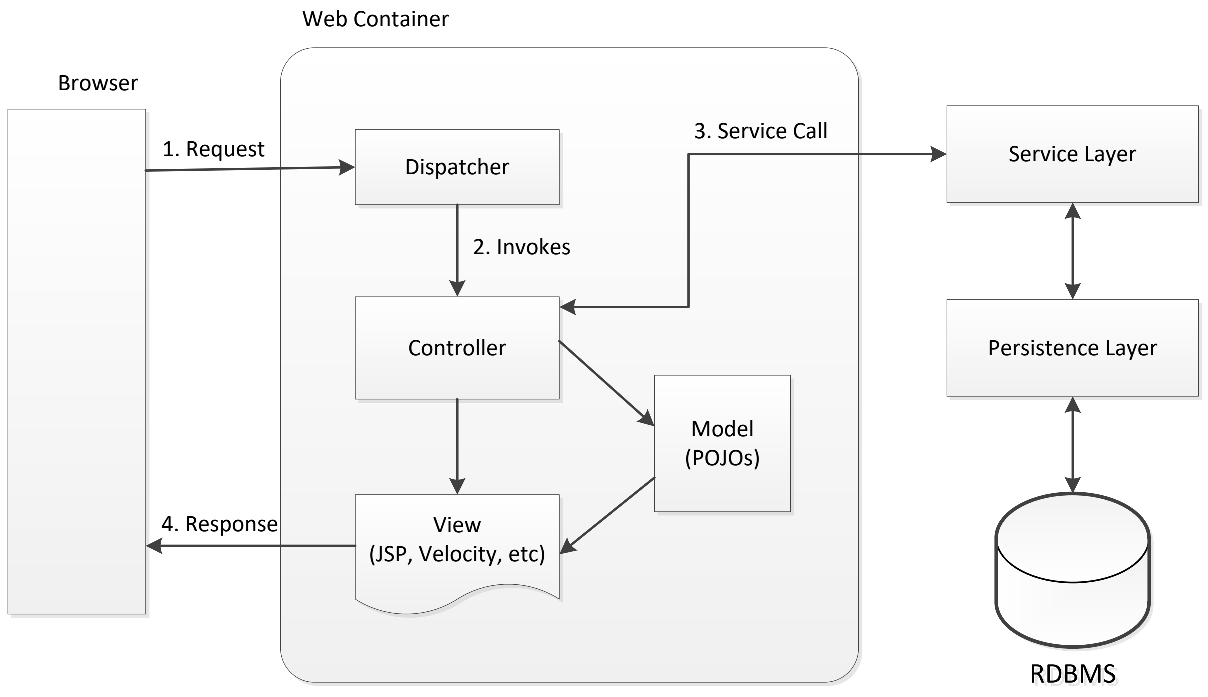

Figure 1 - Model view controller

5.3 Spring Framework

The Spring framework is a comprehensive Java/Java EE application framework designed to address many aspects of Java/Java EE application development. Spring simplify Java development by isolating infrastructural concerns (such as persistence management and transaction management) from domain concerns. The framework handles the former so that developers can focus on the latter. The heart of the Spring framework is a lightweight inversion of control (IoC) container that is able to add enterprise services to simple Java objects declaratively. Spring makes extensive use of aspect-oriented programming (AOP) to provide these services to its components.

In the Spring framework, the Spring MVC module provides comprehensive support for the MVC pattern as illustrated in the previous section. Its primary job is to support the MVC way of dividing application functionality, so it provides explicit support for organizing the web layer into models, views, and controllers. For example, in a Spring MVC application, models usually consist of domain objects that are processed by the service layer and persisted by the persistence layer. Views are usually JSP templates written with Java Standard Tag Library (JSTL) and controllers can be arbitrary Java objects annotated with Spring's controller annotations. This ensures that the separation between the three concerns is clean.

More information on Spring can be found at http://www.springsource.org/.

6 3rd Party Components

6.1 Recommended libraries

Library/Framework	Link to documentation
JavaServer Pages Standard Tag Library (JSTL)	https://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/
Apache POI	https://poi.apache.org/apidocs/
Apache Commons	https://commons.apache.org/proper/commons-lang/apidocs/
Apache log4J	https://logging.apache.org/log4j/2.x/javadoc.html
Ehcache	https://www.ehcache.org/documentation/
Hibernate	https://hibernate.org/orm/documentation/5.4/
JasperReports	https://community.jaspersoft.com/documentation?version=54936
jQuery	https://api.jquery.com/
Java API for XML Web Services (JAX-WS)	https://javaee.github.io/metro-jax-ws/doc/user-guide/release-documentation.html
Jersey	https://eclipse-ee4j.github.io/jersey.github.io/documentation/latest/index.html
1OWASP Enterprise Security API (ESAPI)	https://www.owasp.org/index.php/ESAPI_Documentation

6.1.1 *JavaServer Pages Standard Tag Library (JSTL)*

JSTL is a collection of standard JSP tags that perform common tasks such as controlling the flow structure with conditions and iterators, XML data processing, accessing databases and internationalization. JSTL provides an effective way to embed logic within a JSP page without mixing Java code and XHTML tags and this leads to more maintainable code.

More information on JSTL can be found at https://www.oracle.com/technetwork/java/index-137889.html.

6.1.2 *Apache POI*

Apache POI is a Java library for reading and writing Microsoft Office documents such as Excel, Word and PowerPoint. POI provides the APIs for generating and manipulating Microsoft Office documents dynamically and supports OLE2 files such as XLS, DOC and PPT, as well as the new Office OpenXML format such as XLSX, DOCX and PPTX.

More information on Apache POI can be found at http://poi.apache.org/.

6.1.3 *Apache Commons*

The Apache Commons project is a collection of reusable Java software components. Each Commons project represents a unit of functionality smaller than a full application. The tasks that most of these components undertake are of a general nature and the intention is to help a Java developer build a useful application more quickly by leveraging existing components. There are over 40 components in the Apache Commons so far.

More information on Apache Commons can be found at http://commons.apache.org/.

6.1.4 *Apache log4J*

Apache log4j is a feature-rich, well-designed extendible logging framework, which allows great control over the granularity of logging statements. A main benefit of log4j is that it is highly configurable through external configuration files at runtime. log4j views the logging process in terms of levels of priorities and offers mechanisms to direct logging information to a great variety of destinations, such as a database, file, console, etc.

More information on Apache log4j can be found at http://logging.apache.org/log4j/2.x/.

6.1.5 *Ehcache*

Ehcache is a widely used open source Java distributed cache for general purpose caching, Java EE and light-weight containers. Ehcache provides a rich set of caching capabilities, including pluggable expiration and eviction policies, persistent storage, delivery of change events to your own methods, and even support for distributed caches.

More information on Ehcache can be found at http://ehcache.org/.

6.1.6 *Hibernate*

Hibernate is an object/relational mapping (ORM) framework in which Java objects can be persisted to relational database tables using metadata that describes the mapping between objects and the database. The metadata shields the complexity of dealing directly with SQL and developers can concentrate on developing solutions in terms of business objects. Hibernate implements the Java Persistence API (JPA) defined in the Enterprise JavaBeans (EJB) 3.0 specification.

More information on Hibernate can be found at http://www.hibernate.org/.

6.1.7 *JasperReports*

JasperReports is a Java library designed to aid developers with the task of adding reporting capabilities to Java applications by providing an API to facilitate the ability to generate page-oriented, ready to print documents in a simple and flexible manner. JasperReports can deliver rich content to the screen, printer, or file in PDF, HTML, RTF, XLS, ODT, CSV, or XML format.

More information on JasperReports can be found at https://community.jaspersoft.com/project/jasperreports-library.

6.1.8 *jQuery*

jQuery is a JavaScript library and it consists of a collection of reusable JavaScript code that accomplishes common tasks. It provides many advanced and cross-browser functions that simplify document traversing, event handling, animating, and Ajax interactions for rapid web development.

More information on jQuery can be found at http://jquery.com/.

6.1.9 *Java API for XML Web Services (JAX-WS)*

JAX-WS is a high-level API that simplifies the development of SOAP (Simple Object Access Protocol) web services in the Java platform. JAX-WS utilizes XML messages following the SOAP standard and utilizes a WSDL (Web Services Description Language) file to describe each of the various operations of a particular web service. Clients can use the WSDL file to obtain a proxy to the service. JAX-WS is part of the Java EE platform.

More information on JAX-WS can be found at https://javaee.github.io/metro-jax-ws/.

6.1.10 *Jersey*

The Jersey project is the reference implementation of JAX-RS (Java API for RESTful Web Services). The primary aim of Jersey is to make it easy for developers to build RESTful web services using Java. Jersey is incorporated as a standard part of the Java EE platform.

More information on Jersey can be found at https://jersey.java.net/.

6.1.11 *OWASP Enterprise Security API (ESAPI)*

ESAPI is a free, open source, web application security control library that makes it easier for developers to write lower-risk applications. The ESAPI libraries are designed to make it easier for programmers to retrofit security into existing applications and also serve as a solid foundation for new development.

More information on ESAPI can be found at https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API.

7 Coding Guidelines & Security Considerations

7.1 Java Best Practices and Coding Style

Refer to http://www.javapractices.com for Java coding practices and https://google.github.io/styleguide/javaguide.html for Google Java Style guide.

7.2 Naming and Coding Convention

Refer to https://www.oracle.com/technetwork/java/codeconventions-150003.pdf

7.3 File Upload Guidelines

In the event an application requires file upload feature:

- Whitelist the extensions of the files which can be uploaded.
- If possible, explore the use of APIs which can help us to do a sanity check on file header uploaded to ensure malicious scripts are not uploaded.
- Use GUID as the file name while storing the files.
  Team understands that this requires effort from the developers and can be considered while developing new applications which require the file upload feature
- Do not store files uploaded by the users into the application directory.
- Application shall not keep uploaded files that are no longer in use. Such files must be removed.

7.4 Data Access Coding Guidelines

- Avoid PL/SQL. Put the logics and computation process in the application codes.
- SQL should be optimized, please consult expert advices on complex SQL queries.
- Fields used for search criteria should be indexed especially when querying large chunk of data.
- Use sessions instead of cookies for persistence.
- 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.
- All open connections database must be closed in the finally block of a try-catch statement to prevent connection leak.

- PreparedStatement must be used for variable inputs to be fed in via the corresponding setInt(), setString(), etc methods. See documentation on Java's PreparedStatement.

7.4.1 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.

7.4.2 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.

7.4.3 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.

7.4.4 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

We understand that security is usually considered as a secondary goal compared to functionality, which is to build as many features as possible. However, being aware of security vulnerabilities is critical, and we should minimize security weaknesses as much as possible by practicing secure programming e.g. user input validation.

8 Testing

8.1 Unit testing

Test small bits of your programme e.g. a function call.

- Jshell
- Create a new Client class for testing. See Java's assert.

8.2 Regression Testing

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

8.3 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.

8.4 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.

END OF DOCUMENT