The **Application Development Guidelines** provides guidelines to developers, architects, and technical leads working at DoITT when building applications that expect to be deployed into DoITT infrastructure. They are considered strong recommendations that should be followed. Always use good judgment where appropriate.

Disclaimer

Contents of the DoITT Application Development Wiki are created, edited, and used by DoITT technical teams to guide internal practices. Contents of the Wiki do not constitute policy or standards. Citywide technology policies, guidelines and standards can be found on the DoITT Citywide Policies & Guidelines page on CityShare.

Please note that several links to City of New York resources contained in this document cannot be accessed by external audiences.

Technology

Many DoITT developed applications are deployed to core platforms. Each platform and its supported technologies are listed below. Note that the *www.nyc.gov* and *cityshare.nycnet* environments are deprecated.

NYC.gov (www1.nyc.gov)

- Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64bit Production
- Oracle OHS 12.1.3.0
- WebLogic 11g (10.3.6.0) with Java 1.6 (Note: Some domains have upgraded to Java 1.7)
- WebLogic 12c (12.1.3.0) with Java 1.8

The following domains have been upgraded to 12c on NYC.gov: `events`, `shared`, `cpui`, `captcha`, `marriagebureau`. View the complete list of upgraded NYC.gov Weblogic 12c domains across all environments.

CityShare (cityshare1.nycnet)

- Oracle 12c (WebLogic and OHS) 12.2.1.0.0
- Java 1.8
- Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production

Security

Cybersecurity Requirements for Vendors & Contractors trump all security related standards in this document!

- Never trust user input - cookies count as user input.
- Use NYC.ID for user authentication and self-service account management.
- Use Jasypt to encrypt application properties such as service account passwords. Ensure that the Jasypt key(s) is managed securely as per the established IT Security Policy. (i.e. `JASYPT_CRYPTO_PASSWORD` as a environment variable.)
- Always use TLS for encryption in transit.
  - Non-encrypted (i.e. port 80) traffic on NYC.gov is no longer supported.
  - To secure data in transit, use JDBC with SSL enabled.

```
oracle.net.encryption_client=REQUESTED
oracle.net.encryption_types_client=AES192
oracle.net.crypto_checksum_client=REQUESTED
oracle.net.crypto_checksum_types_client=SHA1
```

- Become familiar with the vulnerabilities in the OWASP Top 10 Project.
- Encrypt Private and Confidential data at rest and in transit.
- To secure data at rest, use Oracle Transparent Data Encryption (TDE), or password-based encryption strategies ought to be used if DoITT Administrators (i.e. DBAs) are not authorized to view the data.
- Use the Java `transient` modifier to prevent serialization of `PRIVATE` and `CONFIDENTIAL` data where appropriate.
- Scrub `PRIVATE` and `CONFIDENTIAL` data before working with production data in lower regions.
- If CAPTCHA is required for a given project, use the DoITT CAPTCHA Service, which wraps Google reCAPTCHA and adds capabilities such as IP throttling.
- Avoid these Top 10 Developer Crypto Mistakes.
- Oracle's Secure Coding Guidelines for the Java Programming Language.

Code Structure, Style and Formatting

- Do not use Hungarian notation.
- Be consistent. Follow a project's pre-existing conventions.
- Format code before committing it to source control.
- Use an automated code inspection tool to find defects. Execute the code inspection tool as part of the build via DoITT's TeamCity CI server.

Building

- Follow DoITT's J2EE Project Structure Guide, which describes the recommended approach for organizing files in a typical IDE workspace such as Eclipse or IntelliJ.
- Follow DoITT's iWay Project Structure Guide for DataShare projects, which describes the recommended approach for organizing files in the workspace of iWay Integration Tools (Eclipse Based iWay IDE).
- Client-side validation is not a substitute for server-side validation. Always perform sever-side validation.
- Ensure that application can be packaged or assembled independently of an IDE (i.e. Ant, Maven, Gradle, etc.) and use DoITT's TeamCity CI server.
  - Read and follow the TeamCity Usage Guide.
- Enable session replication in a clustered environment (i.e. `<persistent-stor--type>replicated_if_clustered</persistent-store-type>` in `weblogic.xml`)
- Specify a `cookie-path` within `weblogic.xml` to avoid cookie collisions.
- Objects that are put into an HTTP session must implement the `java.io.Serializable` interface.
- Take measures to ensure that data cannot be easily corrupted by using normalization techniques (i.e. constraints, foreign keys, indexes, sequences, etc.)
- Avoid using `ThreadLocal`s as they can cause memory leaks.
- Avoid Scriplets in Java Server Pages (JSPs).
  - Encapsulate reusable JSP code in Tag Files.

- Use the correct HTML5 DocType (i.e. `<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">`).
- Close all resources (i.e. Files, Sockets, Connections, etc.) when they are no longer needed.
  - Use `try-with-resources` statement or `finally` block where applicable.
- Do not rely on Akamai, DoITT's CDN, to auto-detect if a page's content is static. Explicitly disable caching on all Web pages that should *NOT* be cached by Akamai. (i.e. Web applications with dynamic content.)
- Develop RESTful APIs. Avoid SOAP.
- Store artifacts (i.e. libraries, certificates, etc.) used by Chef within DoITT's JFrog Artifactory repository management system.

Testing

- Use an established testing framework for unit tests (i.e. JUnit).
- Write integration tests (i.e. tests that use external dependencies such as a database).
- Ideally, test Web UI-related code in an automated way (i.e. Selenium).
- Use Mock JavaMail when testing sending and receiving of emails. This prevents the need for communicating with an SMTP server.
- Test suites must be automatically executed by DoITT's TeamCity CI server.
- Test failure conditions as well as straightforward success and boundary conditions.
- Test serializability (i.e. JUnit-addons)

Deployment

- For applications deployed to a DoITT's core platforms, environment-specific, run-time configurable properties are managed by the Production Support team in the same directory location in the file system per application environment, namely, `/opt/app_config/java_app/<application name or directory tree based on package structure>`. This directory is also added to the CLASSPATH in the WebLogic application server startup scripts. This provides a way for applications to access their properties while providing a way for Production Support team to make changes to them dynamically.
- Do not include references to any server-specific or environment-specific properties in your deployable application. This includes absolute file paths as well.
- Use DoITT Chef automation to configure and deploy software components and applications.

Performance

- Cache static data.
- Disable session management if sessions are not required (i.e. `tracking-enabled` property in `weblogic.xml`).
- Disable CPU intensive logging operations (i.e. Hibernate `show_sql=false`).
- Disable JSP compiler page checking to prevent unnecessary polling of file system for changes to JSP files (i.e. set `page-check-seconds=-1` in `weblogic.xml`).
- Explicitly qualify scope for variables in JSP (i.e. `pageScope`, `requestScope`, `sessionScope`, `applicationScope`)

- Pagination of a result set should be done via SQL, not through application logic. That is, do not fetch large result sets and paginate in memory.
- Do not put large amounts of data in the HTTP Session.
- Create database indexes where appropriate.
- Enable preemptive authentication
- In MS SQL Server, consider using the `with (nolock)` hint.
- Use Dynatrace to troubleshoot performance related issues. Complete the Dynatrace Application Discovery Document prior to deploying your application to production.

Exception Handling

- Do not catch and suppress. Propagate the exception - do not be afraid to begin by always throwing exceptions up to the calling layer, until you discover, usually a business reason, which is the right layer to handle it gracefully.
- Log most exceptions as `error` or `fatal` if necessary. Some may fall under `warn`. Generally they should not be `debug` or `info`.
  - Log exceptions just once.
- If a client can reasonably be expected to recover from an exception, make it a checked exception. If a client cannot do anything to recover from the exception, make it an unchecked exception.
- For unchecked exception, log the stack trace. This is particularly useful when the unchecked exception does not return a meaningful message (i.e. `null` returned from `NullPointerException`).
- Generally speaking, avoid using exceptions for flow control.

Logging

- Use Apache Log4j 2. Do not use Log4j v1.
- Use the guard methods of the form `log.is<Priority>()` to verify that logging should be performed, before incurring the overhead of the logging method call.
- When using the Spring Framework, be sure to include a custom `SimpleMappingExceptionResolver` that logs runtime exceptions as errors. If this is not done then runtime exceptions will **not** be logged making it difficult to troubleshoot.
- Be aware of where the various log files reside on the file system (i.e. application logs in `/opt/logs`).
- Do not log Private or Confidential data.
- Do not use `System.out.println()` or `System.err.println()` or `e.printStackTrace()` ever. Use Log4j 2 instead.
- Ensure logs are viewable using the ELKStack Log Monitoring tool.
  - Use a DoITT supported Log4j 2 pattern:
    - `%d [%t] %-5p %c - %m%n`
    - `%d [%t] %-5p %c - %X{ECID-Context} - %m%n`)
  - Include a filter in your application that logs the Oracle ECID.

Documentation

- When commenting code, describe why the code does what it does, not what it does.
- All technical designs must be written using the AppDevWiki Project Technical Design Template.
  - Store other relevant technical documentation in source control or AppDevWiki as per project needs.
- Understand how to apply and generate a Javadoc where it makes sense. For instance, common modules developed in-house are good candidates for thorough Javadoc documentation.

Source Control

- Use SVN or Git. Do not use CVS.
- Leave meaningful comments when committing code to source control. Describe why the change was committed rather than what was changed.
- Make use of Caliber Requirements Manager identifiers (i.e. DRnnnn), Quality Center defect identifiers (i.e. QCnnnn), or Remedy change request numbers (i.e. CRnnnn) where applicable to more directly associate the reason for the code change.
- Do not commit generated code or artifacts into source control.
- Do not commit commented source code, unless well commented as to why it was commented and when it will be uncommented. See this article on Arstechnica.
- Include Release Notes (i.e. `release_notes.txt`) or a Changelog within source code. Continually update it as defects are fixed and new features are added.

Reporting

Applications must use DoITT's Citywide Performance Reporting (CPR) Oracle Business Intelligence Enterprise Edition (OBIEE) platform for reporting and analysis.

- Light-weight, synchronous reports may be included within Web applications provided requests complete within 5 minutes (under load).
- Asynchronous reports may be included within Web applications provided they do not overload transactional systems (i.e. database). Consider using a job scheduler such as Quartz or the Spring Framework.

Internationalization (i18n) and Localization (l10n)

- Use the UTF-8 character encoding.
  - Use the Oracle `NVARCHAR2` Oracle data type to store UTF-8 strings.
- Create Oracle databases with the UTF-8 character set instead of the default character set. Converting it to UTF-8 later can be difficult.
- Do not put raw English text strings (i.e. sentences, phrases and words) in presentation layer markup. Instead, make judicious use of resource bundles.
  - Use Google Translate for i18n when manual translation is not possible.
- When providing content in multiple languages, automatically detect the user's locale.