

The **Application Development Guidelines** provide information for developers, architects, and technical leads working at DoITT on [NYC.gov](#) and [CityShare](#) Portals. These guidelines are strong recommendations that should be followed. Always use good judgment where appropriate.

Technology

Project teams must familiarize themselves with the core technologies used in each environment.

Security

[Citywide Security Policies](#) trump all security related standards in this document!

- Never trust user input - cookies count as user input.
- Hash and salt passwords – may be deprecated for future projects due to common registration initiatives.
- Use SSL/HTTPS for authentication and any pages where PRIVATE or CONFIDENTIAL data is viewed or entered.
- [OWASP Top 10](#)
 - Avoid Cross-Site Scripting (XSS). HTML-escape all user-controlled data during redisplay in the browser, not during processing of submitted data.
 - Avoid Cross-Site Request Forgeries (XSRF). Each request should have a unique token so that the application can subsequently verify that the request was created from a prior response.
 - Avoid SQL Injection. Use Prepared Statements or APIs/Frameworks that use them. Do not construct SQL statements using string concatenation of unchecked input values.
 - Avoid broken authentication and session management. Invalidate the session during logout and generate a new session upon login.
 - Avoid Insecure Direct Object References. Validate that user supplied input matches a known set of values.
 - Prevent unauthenticated users from accessing arbitrary secure URLs.
 - Avoid non-secure (HTTP) connections when transmitting confidential data. Use HTTPS instead. Persistent login cookies are considered confidential.
 - Avoid unvalidated Redirects and Forwards. Validate that arbitrary redirects and forwards are from trusted domains.
- Encrypt PRIVATE and CONFIDENTIAL data at rest and in transit.
- Scrub PRIVATE and CONFIDENTIAL data before working with Production data.
- If CAPTCHA is applied for a given project, use the [DoITT CAPTCHA Service](#).
- [Secure Coding Guidelines](#) for the Java Programming Language.

Code Structure, Style, and Formatting

AmbySoft Inc. [Coding Standards](#) and [Summary](#) for Java v17.01d

- Do not follow Hungarian notation.
- Be consistent. That is, follow a project's pre-existing conventions.
- Format code before committing it to source control.
- Use an automated code inspection tool to find defects (e.g. [IntelliJ IDEA](#)).

Project Structure

- Read and follow DoITT's [J2EE Project Structure Guide](#).

Building

- Prefer Servlets rather than Portlets.
- Since Spring Portlet MVC is an implementation of the JSR-168 standard, prefer this over raw JSR-168 or other Portlet MVC implementations.

Testing

- Use [JUnit](#) for unit tests.
- Write integration tests (i.e. tests that use external dependencies such as a database).
- Ideally, test Web UI-related code in an automated way.
- Use [Mock JavaMail](#) when testing sending and receiving of emails. This prevents the need for communicating with an SMTP server.
- JUnit test suite must be automated using Ant. The target should then be executed using DoITT's continuous integration server.
- Test failure conditions as well as straightforward success and boundary conditions.
- Follow guidelines in pragmatic [Unit Testing](#) and [Pragmatic Unit Testing: Summary](#) in Java with JUnit.
- Test serializability using [JUnit-addons](#)

Deployment

- Follow the usage guide for DoITT's continuous integration server.
- 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/<project or application name>`. This directory is also added to the CLASSPATH in the 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. This also allows Production Support to utilize a property replacement script to support automatic deployment to upper environments (e.g. TST, STG, PRD, etc).
- For local development, use the property replacement scripts.
- Do not include references to any server-specific or environment-specific properties in your deployable application. This includes absolute file paths as well.
- Enable session replication in `weblogic.xml` (i.e. `persistent-store-type` property)
- Specify a `cookie-path` within `weblogic.xml` to avoid cookie collisions.
- Remember, objects that are put into an HTTP session must implement the `java.io.Serializable` interface.
- When migrating from local to other deployment environments (i.e. DEV, TST, STG, PRD), plan for and gain an understanding of the iPlanet proxy configuration mechanisms to support integration of new web application contexts (i.e. [Syntax and Use of obj.conf](#)).
- To request SSL certificate creation, coordinate with Production Support. Be familiar with the fundamentals of configuring SSL, and be able to distinguish between Keystores for WebLogic, and Vignette.
- Redirect HTTP traffic to HTTPS (for sites that require HTTPS access only)

Performance

- Cache static data.
- Disable CPU intensive logging operation in Production Environments (e.g., Hibernate `show_sql=false`).

- Disable JSP compiler page checking to prevent unnecessary polling of file system for changes to JSP files (e.g. set `page-check-seconds=-1` in `weblogic.xml`).
- Explicitly qualify scope for variables in JSP - `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 HTTP Session.
- Create Database Indexes where appropriate.
- [Improving Hibernate Performance](#)

Data Integrity and Auditing

- Prefer choosing to introduce auditing in requirements gathering and/or design phase, rather than *squeezing it in after the fact* in build/construction phase.
- Consider adding `UPDATE_TIMESTAMP` column to database tables.
- Consider the [DRY principle](#) of software engineering during application design.
- Preserve data model integrity by adding `UNIQUE` and `CHECK` constraints where appropriate.
- Prefer using database `SEQUENCES` where appropriate to generate unique identifiers.
- Enforce business transactions in the application.

Exception Handling

- Do not catch and suppress. Propagate the exception. That is, 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 log level or fatal if necessary. Some may fall under warn. Generally they should not be debug or info.
- 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.
- Log exceptions just once.

Logging

- Use [Apache Log4j](#) for logging.
- 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.
- See log4j [best practices](#).
- When using the Spring Framework, be sure to include a `Custom SimpleMappingExceptionHandlerResolver` that logs runtime exceptions as errors. If this is not done then runtime exceptions are not logged in production when the log level is set to error.
- Be aware of where the various log files reside on the file system and know how to monitor them using “tail” utility (-- in windows using `cygwin+tail`, or `BareTail` a replacement for `WinTail`).
- Do not log `PRIVATE` or `CONFIDENTIAL` data.
- Do not use `System.out.println()` or `System.err.println()` or `e.printStackTrace()` ever. Use log4j instead.

Documentation

- When commenting code, describe why the code does what it does, not what it does.
- Leave meaningful comments when committing code to source control.
- (Legacy) Store technical documentation within source control. All other documentation should be maintained in this Wiki.
- 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

- Add comments on check-in/commit. Always describe why a change is committed as well as what the change is. And consider that what has changed can be discovered using diff.
- Make use of CaliberRM Requirements ID# (DRnnnn) and/or Quality Center Defect ID# and/or CR # where applicable to more directly associate reason for code change commit.
- Do not commit generated code/artifacts into Source Control. In CVS, make judicious use of .cvsignore mechanism.
- 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](#).

Reporting

- Use [Citywide Performance Reporting](#).
- (Legacy) Use Jasper Reports framework.
- For synchronous report generation query request/response must complete in the threshold set by application environment (i.e. currently 10 minutes).
- For sufficiently long running report generation operations, use an asynchronous mechanism to avoid lengthy response wait times for end-users as well as to avoid overloading server resources. Consider using process queuing facility available with Quartz Scheduler utility or [Spring's Task Execution and Scheduling](#).

Internationalization (i18n)

- Prefer to support UTF-8 character encoding. (e.g. NVARCHAR2 Oracle Data Type)
- Create Oracle Databases as UTF-8 from the start rather than other default encoding (e.g. WE8MSWIN1252 on Windows). Not doing so will prove challenging if conversion to UTF-8 is necessary later on.
- Do not put raw English text strings (i.e. sentences, phrases and words) in presentation layer markup. Instead, make judicious use of Java resource bundles along with Google on-demand translation and/or `native2ascii encoding="UTF-8"` pre-processor mechanism.

Preferred APIs and Frameworks

- [JasperReports Library](#)
- [Spring Framework](#)
- [Hibernate](#)
- [EhCache](#) (Consider [OSCache](#) as deprecated.)
- [Lucene \(SOLR\)](#)
- JSR-168 and JSR-286 Portlets - in general, portlet development should be avoided.
- Quartz Scheduler - consider using [Spring Framework](#) instead.
- Web Services – prefer WebLogic or Axis

- [JSTL](#) (instead of Scriptlets)
- [jQuery](#) (instead of straight Javascript)
- [Open Web Application Security Project](#) (OWASP) [Enterprise Security API](#) (ESAPI)

Miscellaneous

- Verify all resources and memory allocated are released when errors occur (i.e. File, Sockets, Database connections, etc.). Use Java `finally` block where applicable.
- Client-side validation is not a substitute for server-side validation.
- Avoid Scriptlets in Java Server Pages (JSPs).
- Use [Tag Files](#) instead of custom tag libraries.
- For JNDI data sources, use the following naming convention:
 - JNDI Name (Value specified in your application): `<ApplicationName>_DS` (e.g. `ELOBBYIST_DS`)
 - Name: `<ApplicationName>_<Database>_JNDI` (e.g. `ELOBBYIST_ORACLE_JNDI`)

Content Delivery Networks

- *Explicitly* disable caching on all pages that should *NOT* be cached by our CDN. Do not rely on the CDN to auto-detect if a page's content is static.
 - [Making sure a web page is not cached, across all browsers](#)
 - [How to set header no cache in spring mvc 3 by annotation](#)