The **Standard Requirements** are DoITT's enterprise wide project requirements. They have been established to improve many aspects of application development such as:

- Reduce project cost by reusing well established, approved, working solutions
- Improve cross team communication
- Set expectations for application features and functionality

Unless otherwise stated, all standard requirements are *MANDATORY*. Note that not all requirements apply to all projects. Deviation from the standard requirements *MUST* be approved by management.

Usability

This category applies to saving and editing data from within an application.

Concurrent Edits

When users are editing the same data we want to protect one user from overwriting the data of another user unintentionally. The solution should be evaluated based on the likelihood of simultaneous access and impact of simultaneous edits.

Preferred Solution

The preferred solution is Optimistic Locking. In this scenario user A opens a record for editing, then user B does the same. User A saves the record, then User B tries to save the same record. The application at this point knows there is a version conflict. The application must display a message to user B that the record was updated and user B cannot save his or her changes.

Alternative Solution

This solution is called Pessimistic Locking. A second user cannot open a record for editing when it is already open by a first user for editing. The application must unlock the record after a period of inactivity.

See also entries for Inactivity - User Interface Timeout and Inactivity - Server Session Timeout.

Multiple User Logins

Multiple logins by the **same** user are generally allowed within web applications for the following reasons:

- If a user closes a browser without logging out, then the user would not be able to login with the same username and password until the old session expired.
- When someone logs in, that person would automatically log off sessions by the same user.
- Users conducting synchronous processes (like time consuming searches) may want to have multiple sessions going so they can work in the application while a synchronous process is executing in another session.

Inactivity - User Interface Timeout

When a user appears to no longer be interacting with an application, the user must be logged out after a maximum of 30 minutes of inactivity. After 25 minutes of inactivity, the application must display a user-friendly session timeout warning in a modal dialog with buttons to *Log Out* or *Stay Logged In*. The *Stay Logged In* button should renew a user's session without a page refresh, typically using AJAX. If no activity takes place within an additional 5 minutes, the user should automatically be logged out *and* the server session terminated.

Some applications may need a shorter time out session due to data sensitivity.

Note: Pop-ups are discouraged due to ad blockers and ADA compliance.

See also Inactivity - Server Session Timeout.

Account Lockout Alert

A user *must* be notified when locked out from authentication due to excessive failed login attempts.

See also Excessive Failed Logins.

Error Messages

This standard refers to application error messages and not validation errors.

Implement user friendly error pages to override default error screens provided by Web and Application Servers. The User Interface Designs should include these pages. Error content *must* include sufficient information to help the user report the issue (e.g. who to contact and how). Error codes may be used to hide unnecessary verbiage from the user, while still providing sufficient information to Portal Support to resolve the issue.

Validation Messages

When users commit errors resulting from the validation of user input (e.g. missing required field or wrong data type), applications must:

- display a message next to the field in question indicating the validation error cause (e.g. required field) *and*
- display a message at the top of the form indicating that an error message appears below.

This requirement allows the users to quickly identify and correct the error. The user interface design and/or error/message inventory *must* also include the location of all validation messages.

Business Forms

Business Forms refer to applications where users 'submit' or 'file' business forms, i.e. DCLA grant applications, Lobbyists' disclosures and COIB financial disclosures. Typical business form functionality must be considered and included in requirements for applications where *needed* by the business.

Typical business form functionality for consideration:

- Whether the application needs to version business forms. Business forms may change due to a variety of reasons such as new laws, policies or the collection of additional data. The requirements must specify how the versions will be managed. For example, if users still need to be able to

update prior submitted versions of the form and resubmit those older versions while the current versions of the forms are also available for submission, requirements must specify how the versions will be managed.
- Managing business form filing periods.
- Functionality pre-populating data from prior form versions to current forms versions
- Preview functionality
- Printer friendly functionality
- eSignatures

See also eSignature and Tab Order.

Tab Order

Ensure that when using the Tab key on forms the tabbing navigation conforms to user expectations.

Pagination

Pagination *must* be used when a large number of records are displayed to the user. The default page size should contain 20 records.

Enter Key

When possible, the *Enter* key should submit the appropriate HTML form. This is an alternative method to submitting the form using a button.

Navigating Without Saving Warning

If a user leaves a page without saving its contents, the application must warn the user that unsaved changes will be lost and allow the user the opportunity to remain on the page.

Asynchronous Processes

When an asynchronous process is started, the application's user interface should indicate to users that the process is running and prevent users from restarting the same process until it has been completed or cancelled. For example, while a report is running, users should not be able to run the *same* report but should be able to run a different report or perform other application functionality.

Synchronous Processes

When a synchronous process is started that will not complete immediately, such as processing a credit card payment, the application's user interface must advise users they must wait until the process is complete to proceed.

Help

All user interface elements that provide assistance to the user such as mouseover tool-tips *must* be specified in the User Interface Flows.

User Name Validation

Applications must strip leading and trailing spaces from user names before performing validation.

Web Browsers and Devices Supported

This standard refers to the browsers and devices with which applications must be compatible.

Applications Used by City Employees

Applications (even public facing) that will be used by city employees must take into account the browsers these employees use and be developed to be compatible with those browsers, at a minimum so the functionality of the application will work.

Applications that are being built for City employees' internal use on the Cityshare environment must be developed to be compatible with their standard web browsers, even if these are not latest versions.

Internal applications requiring remote access must consider additional web browser needs.

Public Facing Applications for Personal Computers

Generally DoITT tries to support browsers which show more than 5% usage in analytics.

For personal computers and similar devices, Public Facing Applications (applications not designed exclusively for use by city employees) should support the following browsers and platforms.

Internet Explorer

- As of December 2013 Internet Explorer 7 and prior versions are **no longer supported**.
- Internet Explorer 8 (Platforms: Vista, XP Pro and Windows 7)
- Internet Explorer 9 (Platforms: Vista and Windows 7)
- Internet Explorer 10 (Platforms: Windows 7 and 8)

Firefox

The latest version of Firefox at the time of development **must** be supported.

Chrome

The latest version of Chrome at the time of development **must** be supported.

Safari

The latest version of Safari at the time of development **may** be supported if needed by the business's users but *this support is not standard*.

Other Browsers

Browsers not listed above are not typically supported.

Security

Database Encryption

Transparent Data Encryption (TDE) is an acceptable solution for handling encryption (and decryption) without losing the ability to search and sort data. For range scans of large data sets, a performance analysis should be done prior to using TDE. There are also some additional limitations, which can be found in Oracle's documentation.

Note: Database replication needs to be tested with TDE. This was mentioned as a problem during the initial attempts of implementing TDE.

See Citywide Data Classification Standard Policy and Citywide Encryption Policy.

Over the Wire Encryption

All private data in transit *must* be transmitted over SSL.

See Citywide Data Classification Standard Policy and Citywide Encryption Policy.

Inactivity - Server Session Timeout

For applications requiring authentication, users must be logged out of their sessions automatically after a maximum of 30 minutes of inactivity with the server. Applications requiring a longer timeout period should seek an exception from Development and Security.

Server session time outs are required even though there are also User Interface Timeouts. If users close browsers without logging out, user interface timeouts will not take place and the server sessions will still be active. To save server resources, timeouts of server sessions are required in addition to user interface timeouts.

Some applications may need a shorter server time out session due to data sensitivity.

See also Inactivity - User Interface Timeout.

Excessive Failed Logins

For public facing City of New York applications, consecutive failed login attempts within a fifteen minute period *must* be handled as follows:

1. After five consecutive failed attempts, CAPTCHA or equivalent functionality *must* be used to verify that a person, and not an automated program, is attempting to log in.
2. After eight consecutive failed attempts (5 without CAPTCHA + 3 with CAPTCHA), the account must be permanently locked until the password is reset by using either of the following self service methods:
   - Email based password reset
   - Challenge-response based password reset using security questions

See also CAPTCHA and Citywide External Identity Management and Password Policy.

Authentication Failure

When authentication fails, a user *must not* be advised which authentication value (email address or password) was incorrect. Doing otherwise allows a malicious user to determine the registered users within applications. An acceptable message is: *The combination of email address and password was not found.*

See also Excessive Failed Logins.

OWASP Top 10

All applications must prevent vulnerabilities defined by the Open Web Application Security Project (OWASP) Top 10 list.

Injection

1. The preferred option is to use a safe API which avoids the use of the interpreter entirely or provides a parameterized interface. Be careful with APIs, such as stored procedures, that are parameterized, but can still introduce injection under the hood.
2. If a parameterized API is not available, you should carefully escape special characters using the specific escape syntax for that interpreter. OWASP's ESAPI provides many of these escaping routines.

Broken Authentication and Session Management

A single set of strong authentication and session management controls. Such controls should strive to:

1. Meet all the authentication and session management requirements defined in OWASP's Application Security Verification Standard (ASVS) areas V2 (Authentication) and V3 (Session Management).
2. Have a simple interface for developers. Consider the ESAPI Authenticator and User APIs as good examples to emulate, use, or build upon.

Cross-Site Scripting (XSS)

1. The preferred option is to properly escape all untrusted data based on the HTML context (body, attribute, JavaScript, CSS, or URL) that the data will be placed into. See the OWASP XSS Prevention Cheat Sheet for details on the required data escaping techniques.
2. Positive or "whitelist" input validation is also recommended as it helps protect against XSS, but is not a complete defense as many applications require special characters in their input. Such validation should, as much as possible, validate the length, characters, format, and business rules on that data before accepting the input.
3. For rich content, consider auto-sanitization libraries like OWASP's AntiSamy or the Java HTML Sanitizer Project.
4. Consider Content Security Policy (CSP) to defend against XSS across your entire site.

Insecure Direct Object References

Preventing insecure direct object references requires selecting an approach for protecting each user accessible object (e.g., object number, filename):

1. **Use per user or session indirect object references.** This prevents attackers from directly targeting unauthorized resources. For example, instead of using the resource's database key, a drop down list of six resources authorized for the current user could use the numbers 1 to 6 to indicate which value the user selected. The application has to map the per-user indirect reference back to the actual database key on the server. OWASP's ESAPI includes both sequential and

random access reference maps that developers can use to eliminate direct object references.
2. **Check access.** Each use of a direct object reference from an untrusted source must include an access control check to ensure the user is authorized for the requested object.

Security Misconfiguration

1. A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Development, QA, and Production environments should all be configured identically (with different passwords used in each environment). This process should be automated to minimize the effort required to setup a new secure environment.
2. A process for keeping abreast of and deploying all new software updates and patches in a timely manner to each deployed environment. This needs to include all code libraries as well.
3. A strong application architecture that provides effective, secure separation between components.
4. Consider running scans and doing audits periodically to help detect future misconfigurations or missing patches.

Sensitive Data Exposure

1. Considering the threats you plan to protect this data from (e.g., insider attack, external user), make sure you encrypt all private data at rest and in transit in a manner that defends against these threats.
2. Don’t store sensitive data unnecessarily. Discard it as soon as possible. Data you don’t have can’t be stolen.
3. Ensure that strong standard algorithms and strong keys are used, and proper key management is in place. Consider using FIPS 140 validated cryptographic modules.
4. Ensure passwords are stored with an algorithm specifically designed for password protection, such as bcrypt, PBKDF2, or scrypt.
5. Disable autocomplete on forms collecting sensitive data and disable caching for pages that contain sensitive data.

Missing Function Level Access Control

Your application should have a consistent and easy to analyze authorization module that is invoked from all of your business functions. Frequently, such protection is provided by one or more components external to the application code.

1. Think about the process for managing entitlements and ensure you can update and audit easily. Don’t hard code.
2. The enforcement mechanism(s) should deny all access by default, requiring explicit grants to specific roles for access to every function.
3. If the function is involved in a workflow, check to make sure the conditions are in the proper state to allow access.

Cross-Site Request Forgery (CSRF)

Preventing CSRF usually requires the inclusion of an unpredictable token in each HTTP request. Such tokens should, at a minimum, be unique per user session.

1. The preferred option is to include the unique token in a hidden field. This causes the value to be sent in the body of the HTTP request, avoiding its inclusion in the URL, which is more prone to exposure.
2. The unique token can also be included in the URL itself, or a URL parameter. However, such placement runs a greater risk that the URL will be exposed to an attacker, thus compromising the secret token.

3. Requiring the user to reauthenticate, or prove they are a user (e.g., via a CAPTCHA) can also protect against CSRF.

Using Components with Known Vulnerabilities

1. Identify all components and the versions you are using, including all dependencies. (e.g., the versions plugin).
2. Monitor the security of these components in public databases, project mailing lists, and security mailing lists, and keep them up to date.
3. Establish security policies governing component use, such as requiring certain software development practices, passing security tests, and acceptable licenses.
4. Where appropriate, consider adding security wrappers around components to disable unused functionality and/ or secure weak or vulnerable aspects of the component.

Unvalidated Redirects and Forwards

1. Simply avoid using redirects and forwards.
2. If used, don't involve user parameters in calculating the destination. This can usually be done.
3. If destination parameters can't be avoided, ensure that the supplied value is valid, and authorized for the user. It is recommended that any such destination parameters be a mapping value, rather than the actual URL or portion of the URL, and that server side code translate this mapping to the target URL. Applications can use ESAPI to override the sendRedirect() method to make sure all redirect destinations are safe.

Temporary Passwords

Temporary passwords should be avoided. When temporary passwords are required, passwords should not contain the characters 1, l, 0, O, X, x, Z, and z to avoid users mistyping them.

See also User Accounts.

Password Expiration

Initial and reset passwords should always be pre-expired. This requirement does not apply to self-registration applications.

See also User Accounts.

Password Storage

Applications *must not* store clear-text or reversibly encrypted passwords. Instead, store the cryptographic hash value of the password. Do not use cryptographically weak hashing algorithms such as MD5.

See also User Accounts.

Password Resets

A password reset *must* send a notification by email to the account owner.

See also User Accounts.

CAPTCHA

On public forms in which a request can be submitted, the form *must* require a CAPTCHA challenge to prevent automated applications from submitting the form. Users must have the option of requesting an audio CAPTCHA challenge per ADA compliance. The DoITT CAPTCHA Service is the recommended approach for including CAPTCHA on a form.

CAPTCHA *must* also be able to be disabled for automated testing.

See also Excessive Failed Logins.

Use SAML for Identity Providers

Applications *must* use the Security Assertion Markup Language (SAML) when communicating with NYC.ID Identity Service Providers.

File Upload Virus Scan

All files uploaded to DoITT servers must be scanned for viruses at the time the file is uploaded.

Web Browser Security Settings

If an agency requires high Web browser security settings for user roles for the application being developed, the application must be developed and tested for the high Web browser security settings.

Auditing and Tracking User Actions

An application *may* require and implement auditing. Requirements should define if auditing is needed and the data changes and user actions in the application trigger and audit. The requirements should also dictate what form of tracking is needed (e.g. message written to log file, database entry, etc.). Private information (e.g. passwords) should never be tracked.

eSignature

Authenticating Electronically Signed Documents

Electronically signed document are electronic documents that comply with the requirements of New York State's Electronic Signature Requirements Act (ESRA). ESRA gives electronic signatures and electronic records used or accepted in New York State the same legal validity and effect as hand-written signatures and paper-based records, subject to certain exceptions stipulated in ESRA. *An electronic document that merely has a means for a user to "agree" to content does not necessarily comply with ESRA.*

DoITT has determined that ESRA requires applications have the following functionality to comply:

1. Signor Identity: Tying the signature to the person which can include enrollment, identification / authentication, chain of evidence
2. Intent to Sign: The signor must be aware that affixing an e-signature carries the same gravitas as a traditional paper signature (ESRA's "ceremony" guidelines).
3. Signature & Document Integrity: Demonstrably protecting the signed document or transaction from tampering. Document Integrity requires
   - a sufficient (SH-256 or greater) hash of the document at the time of signing
   - the storage of the document where it cannot be edited
   - the storage of the document's hash such that it can't be tampered with or replaced

DoITT's current eSignature compliant applications include eLobbyist and COIB EFD (Electronic Filing Disclosure).

Hash Algorithm

Applications that use a hash to guarantee document or file integrity *must* use the SHA-256 algorithm. Stronger algorithms may also be used. Weaker algorithms such as MD5 should be avoided.

Authentic Document Version Store

The authentic version of the document for which a hashed is generated *must* be stored for later retrieval and cannot be edited.

Word documents are not recommended for signed electronic documents; Word updates some meta information inside the file every time the document is opened for reading, not just edited and saved. This changes the file and invalidates the hash. PDFs are convenient for this since they contain a human-readable and printable version of the document, and the PDF itself can be stored in a file system, database or document management system.

Users Accounts

Public Facing User Accounts

All user accounts for public facing applications *must* use the Centralized Public Identity Management System (CPIM) for authentication and account management.

CityShare User Accounts

Users of Internal CityShare applications, whether for a single agency or multiple agencies, *must* authenticate with LDAP using one of the following mechanisms. The first approach is preferred over the second approach. The second approach is preferred over the third approach.

1. City provisioned email address and LAN password
2. Agency, username, and LAN password
3. Employee ID and LAN password

Terms of Service (ToS)

An application that does not use the Centralized Public Identity Management System (CPIM) for account management *must* provide a Terms of Service, which users must accept before using the application. When user information is shared across agency boundaries, the Terms of Service provided by CPIM is insufficient and an additional Terms of Service or EULA may be required.

End User License Agreement (EULA)

Where end user information is to be shared across agency boundaries, the Terms of Use or EULA is required that notifies users what is being done with their information and states that usage of the application constitutes acceptance & authorization from end users.

Privacy

An application which uses the Centralized Public Identity Management System (CPIM) and collects additional personal information *must* have its own privacy statement to cover the use of the additional information collected.

DoITT Password Policy

Where possible leverage the City-wide eDirectory for user account and password policies. Otherwise use DoITT Password Policy.

See DoITT Password Policy.

Administrative Interface

An application *must* contain an administrative interface to manage user authorization (and de-authorization) or to perform other administrative tasks that the business may need to perform regularly such as modifying lists of values or updating locations. Another reason for an Administrative Interface is when multiple users are associated with a single entity or organization.

Note: Any exceptions to this standard must be preapproved by Portal Support and a Development Manager during the requirements phase. Technical Leads cannot approve this exception.

Design

ADA Section 508 Standard Compliance

All Web applications *MUST* comply with the Americans with Disabilities Act (ADA).

- All navigation elements and critical information *MUST* be detectable by a text reader
- All non-structural foreground images *MUST* be labeled with an alt tag descriptor or an empty alt tag if there is no description
- Any use of interactive or dynamic HTML *MUST* be structured so that it appears in the generated code in the context in which it is displayed
- Text and background color combinations *MUST* be high enough contrast to pass the WCAG AA contrast test
- Online forms *MUST* be designed with appropriate label and field attributes

```
Example:
<label  for="name">Your Name</label>
<input  type="text" id="name">
```

- Color *MUST* not be the sole differentiator of any important display item. Use shape, size and position in addition to color. More information: WebAIM on Color-blindness

- Inclusion of a "skip navigation" link at the top of a page that allows users to skip global navigation and jump directly to page content is recommended
- Text-based documents (e.g. HTML, RTF - Rich Text Format) are most compatible with assistive technologies. Provide text-based alternatives to PDFs and graphical diagrams/charts where possible

More information can be found in the Web Content Accessibility Guidelines (WCAG).

Text Size

All sites and applications *must* link to the NYC.gov guidelines for text resizing or provide equivalent (and equivalently maintained) information on a local page.

Screen Resolution and Device Independence

Applications built or redesigned post-October 2013 should be usable and display properly across a full range of mobile and desktop devices from a minimum viewport width of 320 pixels up to a maximum width of 2560 pixels and above.

Websites and applications should be responsive and device-independent to the greatest degree possible.

Brand Elements

Specify any logos, colors, styles, fonts/typography and other visual motifs that must be incorporated into User Interface Designs.

Buttons

All buttons (e.g. Submit, Save, Reset, Cancel, etc.) should be rendered using HTML and CSS, rather than as graphical elements (e.g. png, jpg). This improves the speed at which the web page is rendered and allows content to be edited more easily.

No Graphical Text

All text *must* be rendered using HTML and CSS. Graphics should never be used for text elements outside of highly stylized logo or direct brand elements.

Architecture

Database Access

Java applications *should* useHibernate for database access when anORM is preferred.

Web User Interface Technology

Java applications *must* use Servlets and NOT portlets in their implementation unless the core requirements require portlets. If portlets are required, they should implement the JSR-168 or JSR-286 specification. Proprietary portlets should not be used (e.g., Vignette Portlets).

Transaction Performance

Each transaction should be carefully reviewed and analyzed to determine system and user impact. All transactions that execute longer than the standard response time or add excessive load to the environment should use *throttling techniques*. Some throttling techniques include the following:

1. Have users solve a CAPTCHA
2. Limit the number of threads or transactions
3. Limit the number of threads by a particular user
4. Use a queue, such as MQ Series

Data Transfer

When downloading or extracting large or complex data sets in real-time, the data transfer must take place asynchronously. Session time out would cause the transfer to fail. The requirement *should* specify the data transfer format (e.g., XML, CSV, NEIM).

Deployment

Application Deployment

When deploying to the NYC.gov Re-Arch environment, the automated deployment script *must* be used. Developers should seek to automate deployments through scripting when possible.

Environment Configuration

Environment specific configuration values *must* be managed externally to the application itself. Please see the Application Development Checklist for more information.

Build Artifacts

All build artifacts *must* be published using DoITT's Continuous Integration Server.

Internationalization (i18n) and Localization (l10n)

Character Encoding

The UTF-8 character encoding *must* be used for data persistence, transmission, and viewing. UTF-8 is the dominant character encoding for the World Wide Web.

Language

Applications *must* support the English language. Additional languages can be supported natively or through translation services such as Google Translate.

Performance

CPU Utilization

CPU for all the servers including DB <= 65% plus for DB: no deadlocks, no errors and data should not get truncated. CPU on all the servers should not exceed 85% if test is run for mixed scripts scenario (all the applications together).

Response Time (Web page)

General html page: < 5 seconds

Response Time (File Download)

PDF Forms download: <= 50 seconds

Response Time (Web Service)

Web Services: < 5-10 seconds

Mapping and Location

DoITT's GIS Unit provides services and support for mapping and location-based services.

NYC Interactive Mapping

Interactive mapping applications (for NYC only) can utilize one of DoITT GIS' contracts or services as follows:

- Google Maps API and Maps Engine (cloud-based data store)
- OpenLayers

Interactive Mapping Data

Data for interactive mapping can include DoITT's existing base map and ortho tile cache. A map cache is a collection of pre-rendered map tiles that are used for high performance map display and use. Cached services display quickly because the map image does not have to be rendered on the fly at the time the user/client requests it. DoITT maintains several caches, including multiple ortho caches and a cartographic basemap cache. The ortho caches are based on detailed orthophotography (aerial photographs that have been geometrically corrected to reduce distortion in the image) from several time periods (1924, 1951, 1996, 2006, 2008, 2010 and 2012). The basemap cache displays map features such as roads and buildings that were captured from the most recent orthophotos.

Geospatial Data

Application specific geospatial data should either reside in DoITT's existing spatial repository, be published by the agency in a standards compliant spatial format or when using Google in Maps Engine. Data will either be hosted or accessed from a server publishing the data. e.g. Web Feature Service (WFS), Web Mapping Service (WMS). For specifics, contact the DoITT GIS unit.

NYC Address Validation (i.e., Geocoding)

Address validation and geocoding (for NYC only) should utilize DoITT GIS' Geoclient REST service.

Note: The Geosupporter webservice will be maintained for legacy applications; however all new apps must use the new Geoclient service.

Provide contact information prior to use by sending the information below to gis-development@doitt.nyc.gov. Application (name, purpose, URL); Agency/Department Application Business Owner (name, email, phone); Application Technical Contact (name, email, phone); Estimated application processing requirements e.g., number of transactions per second/minute/hour/day

Valid inputs:

- Address - house number, street, and borough
- Intersection - two cross streets and borough
- Blockface - on street, two cross streets, and borough

- Borough Block and Lot number (BBL)
- Building Identification Number (BIN)

Application Location Awareness

Where applicable, applications should be location aware (i.e., know the geographic coordinates of the location). This is achievable from an address, a BBL, or a BIN using the DoITT GIS Geosupporter web service.

Reports (Formatted and Ad Hoc) and Data Analytics

Applications must use DoITT's analytics solution, CPR (Citywide Performance Reporting), for all reporting and data analytics needs. The analytics solution is OBIEE (Oracle Business Intelligence Enterprise Edition) and includes a business intelligence publisher and reports email delivery. CPR has an Informatica ETL server for data integration and warehousing. This solution should be used instead of custom development in Jasper, using out of the box functionality for COTS products or for cloud based data. Reporting Users will access their reports by the CPR interface. Reporting User training on CPR may be required.

Their are 4 CPR method options depending on the business's Data Classification and reporting needs:

- No private data: Run reports from transactional database if performance is not an issue.
- No private data: Copy the data to a *warehouse* if performance may be an issue such as when there are large data sets.
- Private data: Run reports from the transactional database but use views to protect the private data or do not show private data in Analytics, depending on the encryption techniques.
- Private data: Copy the data to a *warehouse* if performance is an issue, leaving out private data.

See also Web Analytics.

Web Analytics

Data Collection

Web site usage activity should be collected for each application/site that is deployed. Sites using metrics contained defined in DoITT's Web Analytics Metrics will be tagged using the appropriate JavaScript tag. All sites should use the enterprise standard solution. Multiple options are available for tracking, depending on site architecture.

1. Web site usage activity will be collected using a JavaScript tag placed on the application or site pages. Each page should be tested to verify the JS file is included and executes properly and test data isolated from production data. JavaScript tag should be placed near the </body> element on each page of the site or application.
2. Web site usage activity can be collected from web server access logs copied via SCP or SFTP to the web analytics environment.

Site Tracking

The following events should be tracked across a web site, based on business requirements:

- User login/logout from a website
- Process completion/conversion
- Individual content items served dynamically

- Search terms/activity
- Content activity
- Advertisement usage
- Marketing campaigns
- Client/Server error pages

Tracking can be facilitated by explicitly defining events as separate pages within the site or application (e.g. e-mail confirmation, process confirmation, logout, etc.). Additional meta tags, URL parameters, or JavaScript can be added to the pages to track events.

User Sessions

First-party cookies should be used where possible to track users navigating the site. A P3P privacy policy should be used to reduce cookie rejection rates. Third party cookies can be used to track users navigating the site. Tracking users via IP address is feasible, but not recommended. Additional information on the privacy policy can be found here.

Data Availability

Web site usage activity reports should be available within 24 hours of the close of the reporting period. Generally, site usage data is available prior by noon the following day. In some cases, data may be available earlier.

Web Analytics Reports

As an organizational standard, Web Analytics data will be accessed through CPR.

Page Titles

Site/application page titles should include the site or application name and be unique to each particular page. Titles are used within Web Analytics reports to describe page contents. Pages with similar titles are difficult to distinguish. Page titles can be customized using a JavaScript solution in place of HTML title tags.

Page URL

URLs and URL parameters should be structured to allow users to uniquely identify the content on the site. URL aliases should be used to improve discoverability and searchability of content.