Development Guidelines

These are the development/coding guidelines for eZ Publish 5.x kernel, they are the same if you intend to write Bundles, hack on eZ Publish itself or create new functionality for or on top of eZ Publish.

Like most development guidelines these aims to improve security, maintainability, performance and readability of our software. They follow industry standards but sometimes extend them to cater specifically to our needs for eZ Publish ecosystem. The next sections will cover all relevant technologies from a high level point of view.

HTTP

eZ Publish is a web software that is reached via HTTP in most cases, out of the box in eZ Publish 5.x kernel this is specifically: web (usually HTML) or REST.

We aim to follow the latest stable HTTP specification, and industry best practice:

- **Expose our data in a RESTful way**
    - GET, HEAD, OPTIONS & TRACE methods are safe (otherwise known as nullipotent), as in: should never cause changes to resources (note: things like writing a line in a log file are not considered resource changes)
    - PUT & DELETE methods are idempotent, as in multiple identical requests should all have the same result as a single request
    - GET & HEAD methods should be cacheable both on client side, server-side and proxies between, as further defined in the HTTP specification
    - As PUT is for replacing a resource, we should use PATCH in cases where only partial replacement is intended
- **Authenticated traffic**
    - Should use HTTPS
- **Session based traffic**
    - Should follow recommendations for *Authenticated traffic*
    - Should use a per user session CSRF token on all requests using un-safe HTTP methods (POST, PUT, DELETE, PATCH, ...)
    - Should expire session id, session data and CSRF token on login, logout and session time out, except:
        - On login session data from previous session id is moved to new session id, keeping for instance shopping basket on login
    - Should avoid timing attacks by using a random amount of time for login operation
    - Should never use Session id in URI's. And this feature ("SID") must always be disabled on production servers
- **Sessions**
    - Should not be used to store large amounts of data; store data in database and id's in session if needed
    - Should not store critical data: if user deletes his cookies or closes his browser session data is lost
    - Should use an ID generated with enough randomness to prevent prediction or brute-force attacks
- **Cookies (especially session cookies)**
    - Should never store sensitive data in cookies (only exception is session id in session cookie)
    - Should always set *Full domain* to avoid cross-subdomain cooking when on shared domain.
    - Should set *HttpOnly* flag to reduce risk of attacks such as cross-site cooking and cross-site scripting
    - Should set *Secure flag* if HTTPS is used (as recommended above)
    - Must never exceed 4kb
- **Headers**
    - Should never include input data from user input or data from database without sanitizing it
- **Redirects**
    - Should never take url from user input (example: POST parameter), instead allow identifiers instead that are understood by the backend
- **User input**
    - Should always be validated, sanitized, casted and filtered to avoid XSS & clickjacking attacks
        - NB: this includes variables in the php supervariable $_SERVER as well (e.g. hostname should not be trusted)
- **User file uploads**
    - Should follow recommendations for "User input" to validate file name
    - Should place uploaded files in a non public folder to avoid access to execute uploaded file or in case of assets white list the type
    - Should be appropriately limited in size to avoid DOS attacks on disk space, cpu usage by antivirus tool etc...

- **File downloads**
  - Should not rely on user provided file path for non public files, instead use a synthetic id
- **Admin operations**
  - May be placed on a different (sub)domain then the front end website to avoid session stealing across front and backend.
- **Fully support being placed behind a reverse proxy like Varnish**

REST

For now see the living REST v2 specification in our git repository for further details.

UI

eZ Publish is often used as a web content management software, so we always strive to use the HTML/CSS/EcmaScript specifications correctly, and keep new releases up to date on new revisions of those. We furthermore always try to make sure our software gracefully degrades making sure it is useful even on older or less capable web clients (browsers), the industry terms for this approach are:

- Progressive enhancement
- Unobtrusive JavaScript
- Responsive Design

All these terms in general recommends aiming for the minimum standard first, and enhance with additional features/styling if the client is capable of doing so. In essence this allows eZ Publish to be "Mobile first" if the design allows for it, which is recommended. But eZ Publish should always also be fully capable of having different sets of web presentations for different devices using one or several sets of "SiteAccess" (see eZ Publish terms) matching rules for the domain, port or URI, so any kind of device detection can be used together with eZ Publish, making it fully possible to write for instance WAP based websites and interfaces on top of eZ Publish.

WEB Forms/Ajax

As stated in the HTTP section, all unsafe requests to the web server should have a CSRF token to protect against attacks; this includes web forms and ajax requests that don't use the GET http method. As also stated in the HTTP section and further defined in the PHP section, User input should always be validated to avoid XSS issues.

HTML/Templates

All data that comes from backend and in return comes from user input should always be escaped, in case of Twig templates this done by default, but in case of PHP templates, Ajax and other not Twig based output this must be handled manually.

Output escaping must be properly executed according to the desired format, eg. javascript vs. html, but also taking into account the correct character set (see eg. output escaping fallacy when not specifying charset encoding in htmlspecialchars)

Admin

Admin operations that can have a severe impact on the web applications should require providing password and require it again after some time has gone, normally 10 - 20 minutes, on all session based interfaces.

<TODO: Add more coding guidelines for HTML (XHTML5), Javascript, CSS and templates>

PHP

For now see our comprehensive coding standard & guidelines wiki page on github.

> **eZ Coding Standards Tools**
> See also eZ Coding Standards Tools repository to get the configuration files for your favorite tools.

Public API

The PHP Public API provided in eZ Publish 5.0 is in most cases in charge of checking permissions to data for you, but some API's are not documented to throw UnauthorizedException, which means that it is the consumer of the API's who is responsible for checking permissions.

The following example shows how this is done in the case of loading users:

loadUser()

```
// Get a user
$userId = (int)$params['id'];
$userService = $repository->getUserService();
$user = $userService->loadUser( $userId );

// Now check that current user has access to read this user
if ( !$repository->canUser( 'content', 'read', $user ) )
{
    // Generates message: User does not have access to 'content' 'read' with id '10'
    throw new \eZ\Publish\Core\Base\Exceptions\UnauthorizedException( 'content',
'read', array( 'id' => $userId ) );
}
```

Command line

Output must always be escaped when displaying data from the database.

<TODO: Expand on how best practice is to handle user input in eZ Publish 5 to avoid XSS issues>

Data & Databases

- Values coming from variables should always be appropriately quoted or binded in SQL statements
- The SQL statements used should never be created by hand with one version per supported database, as this increases both the maintenance load and the chances for security-related problems
- Usage of temporary tables is discouraged, as their behaviour is very different on different databases. Subselects should be prefererred (esp. since recent mysql versions have much better support for them)
- Full table locking is discouraged

<TODO: guidelines for how data should be stored for maximum portability (hint: XML & abstraction)>

Sessions

- Business logic should not depend on database connections being either persistent or not persistent
- The connection to the database should always be opened as late as possible during page execution. Ideally, to improve scalability, a web page executing no queries should not connect to the db at all (note that closing the db connection as soon as possible is a tricky problem, as we expect to support persistent db connections as well for absolute best performances)
- The same principle applies to configurations where a master/slave db setup is in use: the chance for a failure due to a database malfunction should not increase with the number of db servers at play, but actually decrease
- It is recommended to avoid as much as possible statements which alter the current session, as they slow down the application, are brittle and hard to debug.
  Point in case; if a db session locks a table then is abruptly terminated, the table might stay locked for a long time

Transactions

- Transactions should always be used to wrap sql statements which affect data in multiple tables: either all data changes go through or none of them
- Transactions are prone to locking issues, so the code executed within a transaction should be limited to the minimum necessary amount (ex. clearing caches should be done after the transaction is committed)
- When using transactions, always consider side effects on external system, such as on-disk storage. F.e. is a transaction relative to creating an image variation is rolled back, the corresponding file should not be left on disk
- Nested transactions are supported in the following way:
    - a transaction within another one will not commit when requested, only the outhermost transaction will commit
    - a transaction within another one will roll back all the way to the start of the outhermost transaction when requested
    - as a result a transaction shall never be rolled back just as a means of cancelling its work - the side effect might be of cancelling other work which had just been done previously

Limitations in the SQL dialect supported

Striving to support Mysql 5, PostgreSQL xx and Oracle 10, the following limitations apply:

- Tables, columns and other db objects should not use names longer than 30 chars
- Varchar columns with a definition of *default "" not null* are discouraged
- For SELECTs, offset and limit have to be handled by the php layer, not hardcoded in the sql
- Never treat a NULL varchar value as semantically different from an empty string value
- The select list of a query cannot contain the same field multiple times
- For GROUP BY statements, all fields in the group by clause should be in the select list as well
- For SELECTs, usage of the AS token is allowed in the select list, but not in the list of tables
- Do not put quotes around numeric values (use proper casting/escaping to avoid SQL injection)
- *<TODO: finish sql guidelines>*