Context aware HTTP cache

Use case

As it is based on Symfony 2, eZ Platform uses HTTP cache extended with features like content awareness. However, this cache management is only available for anonymous users due to HTTP restrictions.

It is of course possible to make HTTP cache vary thanks to the `Vary` response header, but this header can only be based on one of the request headers (e.g. `Accept-Encoding`). Thus, to make the cache vary on a specific context *(for example a hash based on a user roles and limitations)*, this context must be present in the original request.

Feature

As the response can vary on a request header, the base solution is to make the kernel do a sub-request in order to retrieve the user context hash (aka **user hash**). Once the *user hash* has been retrieved, it's injected in the original request in the `X-User-Hash` custom header, making it possible to *vary* the HTTP response on this header:

```
<?php
use Symfony\Component\HttpFoundation\Response;

// ...

// Inside a controller action
$response = new Response();
$response->setVary('X-User-Hash');
```

This solution is implemented in Symfony reverse proxy (aka *HttpCache*) and is also accessible to dedicated reverse proxies like Varnish.

Note that sharing ESIs across SiteAccesses is not possible by design (see **EZP-22535** - Cached ESI can not be shared across pages/siteaccesses due to "pathinfo" property **CLOSED** for technical details)

Vary by User
In cases where you need to deliver content uniquely to a given user, and tricks like using JavaScript and cookie values, hinclude, or disabling cache is not an option. Then remaining option is to vary response by cookie:

```
$response->setVary('Cookie');
```

Unfortunately this is not optimal as it will by default vary by all cookies, including those set by add trackers, analytics tools, recommendation services, etc. However, as long as *your* application backend does not need these cookies, you can solve this by stripping everything but the session cookie. Example for Varnish can be found in the default VCL examples in part dealing with User Hash, for single-server setup this can easily be accomplished in Apache / Nginx as well.

HTTP cache clear

As eZ Platform uses FOSHttpCacheBundle, this impacts the following features:

- HTTP cache purge
- User context hash

Varnish proxy client from FOSHttpCache lib is used for clearing eZ HTTP cache, even when using Symfony HttpCache. A single `BAN` request is sent to registered purge servers, containing a `X-Location-Id` header. This header contains all Location IDs for which objects in cache need to be cleared.

Symfony reverse proxy

Symfony reverse proxy (aka HttpCache) is supported out of the box, all you have to do is to activate it.

Varnish

Please refer to Using Varnish

User context hash

FOSHttpCacheBundle *User Context feature* is activated by default.

As the response can vary on a request header, the base solution is to make the kernel do a sub-request in order to retrieve the context (aka **user context hash**). Once the *user hash* has been retrieved, it's injected in the original request in the `X-User-Hash` header, making it possible to *vary* the HTTP response on this header:

Name of the user hash header is configurable in FOSHttpCacheBundle. By default eZ Platform sets it to `**X-User-Hash**`.

```
<?php
use Symfony\Component\HttpFoundation\Response;

// ...

// Inside a controller action
$response = new Response();
$response->setVary('X-User-Hash');
```

This solution is implemented in Symfony reverse proxy (aka *HttpCache*) and is also accessible to dedicated reverse proxies like Varnish.

Workflow

Please refer to FOSHttpCacheBundle documentation on how user context feature works.

User hash generation

Please refer to FOSHttpCacheBundle documentation on how user hashes are being generated.

eZ Platform already interferes with the hash generation process by adding current user permissions and limitations. You can also interfere in this process by implementing custom context provider(s).

User hash generation with Varnish 3

The behavior described here comes out of the box with Symfony reverse proxy, but it's of course possible to use Varnish to achieve the same.

```
# Varnish 3 style for eZ Platform
# Our Backend - We assume that eZ Platform Web server listen on port 80 on the same
machine.
backend ezplatform {
    .host = "127.0.0.1";
    .port = "80";
}

# Called at the beginning of a request, after the complete request has been received
sub vcl_recv {

    # Set the backend
    set req.backend = ezplatform;

    # ...

    # Retrieve client user hash and add it to the forwarded request.
    call ez_user_hash;

    # If it passes all these tests, do a lookup anyway;
    return (lookup);
}

# Sub-routine to get client user hash, for context-aware HTTP cache.
# Don't forget to correctly set the backend host for the Curl sub-request.
sub ez_user_hash {

    # Prevent tampering attacks on the hash mechanism
    if (req.restarts == 0
        && (req.http.accept ~ "application/vnd.fos.user-context-hash"
            || req.http.x-user-context-hash
        )
    ) {
        error 400;
    }

    if (req.restarts == 0 && (req.request == "GET" || req.request == "HEAD")) {
        # Get User (Context) hash, for varying cache by what user has access to.
        # https://doc.ez.no/display/EZP/Context+aware+HTTP+cach

        # Anonymous user w/o session => Use hardcoded anonymous hash to avoid backend
lookup for hash
        if (req.http.Cookie !~ "eZSESSID" && !req.http.authorization) {
            # You may update this hash with the actual one for anonymous user
            # to get a better cache hit ratio across anonymous users.
            # Note: Then needs update every time anonymous user role assignments
change.
            set req.http.X-User-Hash = "38015b703d82206ebc01d17a39c727e5";
        }
        # Pre-authenticate request to get shared cache, even when authenticated
        else {
            set req.http.x-fos-original-url    = req.url;
            set req.http.x-fos-original-accept = req.http.accept;
            set req.http.x-fos-original-cookie = req.http.cookie;
            # Clean up cookie for the hash request to only keep session cookie, as
hash cache will vary on cookie.
            set req.http.cookie = ";" + req.http.cookie;
            set req.http.cookie = regsuball(req.http.cookie, "; +", ";");
            set req.http.cookie = regsuball(req.http.cookie, ";(eZSESSID[^=]*)=", ";
\1=");
            set req.http.cookie = regsuball(req.http.cookie, ";[^ ][^;]*", "");
            set req.http.cookie = regsuball(req.http.cookie, "^[; ]+|[; ]+$", "");

            set req.http.accept = "application/vnd.fos.user-context-hash";
            set req.url = "/_fos_user_context_hash";

            # Force the lookup, the backend must tell not to cache or vary on all
            # headers that are used to build the hash.

            return (lookup);
        }
    }

    # Rebuild the original request which now has the hash.
    if (req.restarts > 0
        && req.http.accept == "application/vnd.fos.user-context-hash"
    ) {
        set req.url         = req.http.x-fos-original-url;
        set req.http.accept = req.http.x-fos-original-accept;
        set req.http.cookie = req.http.x-fos-original-cookie;

        unset req.http.x-fos-original-url;
        unset req.http.x-fos-original-accept;
        unset req.http.x-fos-original-cookie;

        # Force the lookup, the backend must tell not to cache or vary on the
        # user hash to properly separate cached data.

        return (lookup);
    }
}

sub vcl_fetch {

    # ...

    if (req.restarts == 0
        && req.http.accept ~ "application/vnd.fos.user-context-hash"
        && beresp.status >= 500
    ) {
        error 503 "Hash error";
    }
}

sub vcl_deliver {
```

```
# On receiving the hash response, copy the hash header to the original
# request and restart.
if (req.restarts == 0
    && resp.http.content-type ~ "application/vnd.fos.user-context-hash"
    && resp.status == 200
) {
    set req.http.x-user-hash = resp.http.x-user-hash;

    return (restart);
}

# If we get here, this is a real response that gets sent to the client.

# Remove the vary on context user hash, this is nothing public. Keep all
# other vary headers.
set resp.http.Vary = regsub(resp.http.Vary, "(?i),? *x-user-hash *", "");
set resp.http.Vary = regsub(resp.http.Vary, "^, *", "");
if (resp.http.Vary == "") {
    remove resp.http.Vary;
}

# Sanity check to prevent ever exposing the hash to a client.
```

```
    remove resp.http.x-user-hash;
}
```

User hash generation with Varnish 4

```
// Varnish 4 style for eZ Platform
// Complete VCL example

vcl 4.0;

// Our Backend - Assuming that web server is listening on port 80
// Replace the host to fit your setup
backend ezplatform {
    .host = "127.0.0.1";
    .port = "80";
}

// Called at the beginning of a request, after the complete request has been received
sub vcl_recv {

    // Set the backend
    set req.backend_hint = ezplatform;

    // ...

    // Retrieve client user hash and add it to the forwarded request.
    call ez_user_hash;

    // If it passes all these tests, do a lookup anyway.
    return (hash);
}

// Called when the requested object has been retrieved from the backend
sub vcl_backend_response {
    if (bereq.http.accept ~ "application/vnd.fos.user-context-hash"
        && beresp.status >= 500
    ) {
        return (abandon);
    }

    // ...
}

// Sub-routine to get client user hash, for context-aware HTTP cache.
sub ez_user_hash {

    // Prevent tampering attacks on the hash mechanism
    if (req.restarts == 0
        && (req.http.accept ~ "application/vnd.fos.user-context-hash"
            || req.http.x-user-hash
        )
    ) {
        return (synth(400));
    }

    if (req.restarts == 0 && (req.method == "GET" || req.method == "HEAD")) {
        // Get User (Context) hash, for varying cache by what user has access to.
        // https://doc.ez.no/display/EZP/Context+aware+HTTP+cache

        // Anonymous user w/o session => Use hardcoded anonymous hash to avoid backend
lookup for hash
        if (req.http.Cookie !~ "eZSESSID" && !req.http.authorization) {
            // You may update this hash with the actual one for anonymous user
            // to get a better cache hit ratio across anonymous users.
            // Note: You should then update it every time anonymous user rights
change.
            set req.http.X-User-Hash = "38015b703d82206ebc01d17a39c727e5";
        }
        // Pre-authenticate request to get shared cache, even when authenticated
        else {
            set req.http.x-fos-original-url    = req.url;
            set req.http.x-fos-original-accept = req.http.accept;
            set req.http.x-fos-original-cookie = req.http.cookie;
            // Clean up cookie for the hash request to only keep session cookie, as
hash cache will vary on cookie.
            set req.http.cookie = ";" + req.http.cookie;
            set req.http.cookie = regsuball(req.http.cookie, "; +", ";");
            set req.http.cookie = regsuball(req.http.cookie, ";(eZSESSID[^=]*)=", ";
\1=");
            set req.http.cookie = regsuball(req.http.cookie, ";[^ ][^;]*", "");
            set req.http.cookie = regsuball(req.http.cookie, "^[; ]+|[; ]+$", "");
            set req.http.accept = "application/vnd.fos.user-context-hash";
            set req.url = "/_fos_user_context_hash";

            // Force the lookup, the backend must tell how to cache/vary response
containing the user hash
            return (hash);
        }
    }

    // Rebuild the original request which now has the hash.
    if (req.restarts > 0
        && req.http.accept == "application/vnd.fos.user-context-hash"
    ) {
        set req.url         = req.http.x-fos-original-url;
        set req.http.accept = req.http.x-fos-original-accept;
        set req.http.cookie = req.http.x-fos-original-cookie;
        unset req.http.x-fos-original-url;
        unset req.http.x-fos-original-accept;
        unset req.http.x-fos-original-cookie;

        // Force the lookup, the backend must tell not to cache or vary on the
        // user hash to properly separate cached data.

        return (hash);
    }
}

sub vcl_deliver {
    // On receiving the hash response, copy the hash header to the original
    // request and restart.
    if (req.restarts == 0
        && resp.http.content-type ~ "application/vnd.fos.user-context-hash"
    ) {
        set req.http.x-user-hash = resp.http.x-user-hash;
        return (restart);
    }

    // If we get here, this is a real response that gets sent to the client.
    // Remove the vary on context user hash, this is nothing public. Keep all
    // other vary headers.
    set resp.http.Vary = regsub(resp.http.Vary, "(?i),? *x-user-hash *", "");
    set resp.http.Vary = regsub(resp.http.Vary, "^, *", "");
    if (resp.http.Vary == "") {
        unset resp.http.Vary;
    }

    // Sanity check to prevent ever exposing the hash to a client.
    unset resp.http.x-user-hash;
    if (client.ip ~ debuggers) {
        if (obj.hits > 0) {
            set resp.http.X-Cache = "HIT";
            set resp.http.X-Cache-Hits = obj.hits;
        } else {
            set resp.http.X-Cache = "MISS";
        }
    }
}
```

Default options for FOSHttpCacheBundle defined in eZ

The following configuration is defined in eZ by default for FOSHttpCacheBundle. You may override these settings.

```
fos_http_cache:
    proxy_client:
        # "varnish" is used, even when using Symfony HttpCache.
        default: varnish
        varnish:
            # Means http_cache.purge_servers defined for current SiteAccess.
            servers: [$http_cache.purge_servers$]

    user_context:
        enabled: true
        # User context hash is cached during 10min
        hash_cache_ttl: 600
        user_hash_header: X-User-Hash
```

Credits

This feature is based on Context aware HTTP caching post by asm89.