View decorators

Django provides several decorators that can be applied to views to support various HTTP features.

See Decorating the class for how to use these decorators with class-based views.

Allowed HTTP methods

The decorators in `django.views.decorators.http` can be used to restrict access to views based on the request method. These decorators will return a `django.http.HttpResponseNotAllowed` if the conditions are not met.

`require_http_methods(request_method_list)` [source]

Decorator to require that a view only accepts particular request methods. Usage:

```
from django.views.decorators.http import require_http_methods


@require_http_methods(["GET", "POST"])
def my_view(request):
    # I can assume now that only GET or POST requests make it this far
    # ...
    pass
```

Note that request methods should be in uppercase.

Changed in Django 5.0:
Support for wrapping asynchronous view functions was added.

`require_GET()`

Decorator to require that a view only accepts the GET method.

Changed in Django 5.0:
Support for wrapping asynchronous view functions was added.

`require_POST()`

Decorator to require that a view only accepts the POST method.

Changed in Django 5.0:
Support for wrapping asynchronous view functions was added.

`require_safe()`

Decorator to require that a view only accepts the GET and HEAD methods. These methods are commonly considered "safe" because they should not have the significance of taking an action other than retrieving the requested resource.

Note: Web servers should automatically strip the content of responses to HEAD requests while leaving the headers unchanged, so you may handle HEAD requests exactly like GET requests in your views. Since some software, such as link checkers, rely on HEAD requests, you might prefer using `require_safe` instead of `require_GET`.

Changed in Django 5.0:
Support for wrapping asynchronous view functions was added.

Conditional view processing

The following decorators in `django.views.decorators.http` can be used to control caching behavior on particular views.

`condition(etag_func=None, last_modified_func=None)` [source]

`etag(etag_func)` [source]

`last_modified(last_modified_func)` [source]

These decorators can be used to generate `ETag` and `Last-Modified` headers; see conditional view processing.

Changed in Django 5.0:
Support for wrapping asynchronous view functions was added.

GZip compression

The decorators in `django.views.decorators.gzip` control content compression on a per-view basis.

`gzip_page()`

This decorator compresses content if the browser allows gzip compression. It sets the `Vary` header accordingly, so that caches will base their storage on the `Accept-Encoding` header.

Changed in Django 5.0:
Support for wrapping asynchronous view functions was added.

Vary headers

The decorators in `django.views.decorators.vary` can be used to control caching based on specific request headers.

`vary_on_cookie(func)`

Changed in Django 5.0:
Support for wrapping asynchronous view functions was added.

`vary_on_headers(*headers)` [source]

The `Vary` header defines which request headers a cache mechanism should take into account when building its cache key.

See using vary headers.

Changed in Django 5.0:
Support for wrapping asynchronous view functions was added.

Caching

The decorators in `django.views.decorators.cache` control server and client-side caching.

`cache_control(**kwargs)` [source]

This decorator patches the response's `Cache-Control` header by adding all of the keyword arguments to it. See `patch_cache_control()` for the details of the transformation.

Changed in Django 5.0:
Support for wrapping asynchronous view functions was added.

`never_cache(view_func)` [source]

This decorator adds an `Expires` header to the current date/time.

This decorator adds a `Cache-Control: max-age=0, no-cache, no-store, must-revalidate, private` header to a response to indicate that a page should never be cached.

Each header is only added if it isn't already set.

Changed in Django 5.0:

Support for wrapping asynchronous view functions was added.

Common

The decorators in `django.views.decorators.common` allow per-view customization of `CommonMiddleware` behavior.

`no_append_slash()` [source]

This decorator allows individual views to be excluded from `APPEND_SLASH` URL normalization.

Changed in Django 5.0:

Support for wrapping asynchronous view functions was added.