Middleware

August 17, 2026

Middleware

Overview

Middleware in this directory is made of factories that return handlers operating on Hono's request `Context` and its downstream `next` continuation. Some act before calling `next`, such as authentication, request-body limiting, language selection, request IDs, and method rewriting; others inspect or modify the completed response after `next` returns, such as compression, ETags, cache storage, security headers, logging, timing, and URL canonicalization. [src/middleware/basic-auth/index.ts:105-152; src/middleware/compress/index.ts:88-135; src/middleware/etag/index.ts:84-131; src/middleware/secure-headers/secure-headers.ts:221-234]

The central data path is therefore a mutable context: middleware reads request method, URL, headers, body, route information, and runtime execution context; it may set context variables, replace `c.req.raw`, replace `c.res`, return a response early, or throw an `HTTPException`. [src/middleware/body-limit/index.ts:61-109; src/middleware/language/language.ts:304-307; src/middleware/request-id/request-id.ts:46-58; src/middleware/method-not-allowed/index.ts:64-140]

The part contains middleware intended for cross-cutting HTTP concerns rather than application controllers. Its exported handlers are attached by the application or router, and their effect depends on registration order because many surround the downstream chain with `await next()`. [src/middleware/cors/index.ts:96-156; src/middleware/cache/index.ts:266-323; src/middleware/powered-by/index.ts:30-34]

How it works

A typical request can first be rejected or transformed before route handling. `bodyLimit` passes requests without a body through, compares a trustworthy `Content-Length` against the configured limit when present, or reads an unbounded or transfer-encoded stream into chunks, rejects once the accumulated size exceeds the limit, and

reconstructs `c.req.raw` before continuing. [src/middleware/body-limit/index.ts:61-109] `csrf` only examines unsafe form-style requests; it continues when either the configured `Sec-Fetch-Site` policy or the configured origin policy accepts the request, otherwise it throws a forbidden response. [src/middleware/csrf/index.ts:94-150]

Authentication middleware follows the same gate pattern. Basic authentication parses the incoming credentials, either invokes `verifyUser` or compares against configured users with `timingSafeEqual`, invokes an optional success callback, and only then calls `next`; otherwise it throws an unauthorized response with a `WWW-Authenticate` challenge. [src/middleware/basic-auth/index.ts:105-152] Bearer authentication reads a configurable header, validates its scheme and token syntax, checks the token through a verifier or timing-safe comparison, and distinguishes malformed credentials from an invalid token through separate HTTP exceptions. [src/middleware/bearer-auth/index.ts:156-220]

JWT and JWK middleware similarly read a bearer token from a configurable header, with an optional cookie fallback. JWT verifies against the configured secret and algorithm, while JWK verification can combine supplied keys with keys fetched from a configured JWKS location; both place a verified payload in `jwtPayload` before continuing. [src/middleware/jwt/jwt.ts:78-161; src/middleware/jwk/jwk.ts:78-172] JWK authentication may allow a missing token when `allow_anon` is set, but malformed credentials and missing tokens otherwise result in unauthorized responses. [src/middleware/jwk/jwk.ts:81-135]

Request-scoped state is added directly to the context. `requestId` accepts a valid inbound identifier or generates one, stores it as `requestId`, and emits it in the configured response header. [src/middleware/request-id/request-id.ts:41-58] `languageDetector` merges caller options with defaults, validates them, tries configured detectors in order, optionally caches a found language in a cookie, writes the final or fallback language to `language`, and then invokes the downstream handler. [src/middleware/language/language.ts:221-267; src/middleware/language/language.ts:292-307] In Node environments, `contextStorage` runs the downstream continuation in `AsyncLocalStorage`, allowing code later in that asynchronous chain to retrieve the current context. [src/middleware/context-storage/index.ts:6-10; src/middleware/context-storage/index.ts:43-58]

The downstream response is then available to wrapping middleware. `compress` checks response status, existing encodings, request method, content length, content type, and `Cache-Control`; if compression is applicable, it adds `Vary: Accept-Encoding`, negotiates a supported content encoding from the request, pipes the body through

`CompressionStream`, removes `Content-Length`, and weakens a strong ETag. [src/middleware/compress/index.ts:88-135] `etag` applies only to successful GET, HEAD, or QUERY responses, accepts an existing ETag or derives one from a cloned response body, and replaces a matching response with a bodyless not-modified response containing retained headers. [src/middleware/etag/index.ts:79-131]

Cache middleware first bypasses cache lookup for methods other than GET or QUERY and for requests carrying `Authorization`. For eligible requests it builds a cache key from the request URL or custom key, includes a content digest and representation metadata for QUERY requests, incorporates configured `Vary` request-header values, and returns a cache match immediately. [src/middleware/cache/index.ts:95-152; src/middleware/cache/index.ts:266-304] After a miss, it only stores configured cacheable statuses and declines storage when response `Vary` is incompatible with configuration, cache-control prevents caching, or the response sets a cookie. [src/middleware/cache/index.ts:72-80; src/middleware/cache/index.ts:306-322]

Composition helpers alter this control flow. `some` tries handlers in order, proceeding after the first non-failing result and rethrowing the most recent error if none succeeds. [src/middleware/combine/index.ts:38-68] `every` composes every supplied handler while treating a `false` result as an unmet condition, and `except` turns path patterns and predicate conditions into an exclusion around that composition. [src/middleware/combine/index.ts:99-116; src/middleware/combine/index.ts:141-165]

API surface

The HTTP surface is middleware-oriented: its handlers are registered on application routes rather than acting as standalone controllers. The parsed surface records that most declared endpoints have no guard or middleware of their own, which is consistent with these factories being attached around application handlers rather than replacing them. [parsed surface]

Authentication factories validate required setup at construction time. `basicAuth` requires either credential fields or `verifyUser`; `bearerAuth` requires configured token material or `verifyToken`; JWT requires a secret and algorithm; and JWK requires keys or a JWKS location plus Web Crypto key import support. [src/middleware/basic-auth/index.ts:80-103; src/middleware/bearer-auth/index.ts:104-120; src/middleware/jwt/jwt.ts:54-76; src/middleware/jwk/jwk.ts:49-76] Authentication failures return challenges with request-specific or configured realm data, while a failed JWK verification can rethrow an ordinary `Error` originating from key retrieval or

verification rather than converting it to an unauthorized response. [src/middleware/jwt/jwt.ts:145-184; src/middleware/jwk/jwk.ts:153-195]

CORS accepts fixed, list-based, or callback-based origin and method policies. It sets origin, credential, and exposed-header fields before proceeding, but answers OPTIONS directly with an empty response after setting allowed methods, requested or configured headers, optional maximum age, and relevant `Vary` fields. [src/middleware/cors/index.ts:63-94; src/middleware/cors/index.ts:96-156]

IP restriction accepts a connection-info function or direct address function and supports wildcard, literal-address, CIDR, and predicate rules. Deny rules take precedence; an allow-list permits only matches when nonempty; absent or invalid client addresses become forbidden responses unless a custom error handler supplies the response. [src/middleware/ip-restriction/index.ts:51-166; src/middleware/ip-restriction/index.ts:218-278]

Method handling can re-dispatch a reconstructed request through the supplied application. `methodOverride` ignores GET, then reads the override from a form field, header, or query parameter, removes the source value where applicable, and calls `app.fetch` with the revised method. [src/middleware/method-override/index.ts:60-138] `methodNotAllowed` waits for a downstream not-found response, builds a route-method matcher from the application's routes, and replaces the response with a method-not-allowed response and `Allow` header when the path exists but the requested method is unsupported. [src/middleware/method-not-allowed/index.ts:64-140]

Static serving is deliberately runtime-adapter-facing: callers supply `getContent` and may supply path joining and directory inspection. It rejects undecodable or traversal-like request paths by invoking `onNotFound` and continuing; for found data it sets a MIME type, can select a precompressed representation from `Accept-Encoding`, invokes `onFound`, and returns the body. [src/middleware/serve-static/index.ts:13-21; src/middleware/serve-static/index.ts:35-48; src/middleware/serve-static/index.ts:54-125]

Response presentation middleware changes completed responses. `prettyJSON` formats JSON when a configured query parameter is present or formatting is forced. [src/middleware/pretty-json/index.ts:46-55] `poweredBy` sets `X-Powered-By` after downstream handling, whereas `secureHeaders` sets its selected headers after downstream handling and can remove that header. [src/middleware/powered-by/index.ts:30-34; src/middleware/secure-headers/secure-headers.ts:179-234] JSX rendering installs a context renderer and layout; its renderer either returns HTML or

produces a readable stream with caller-selected stream headers. [src/middleware/jsx-renderer/index.ts:31-78; src/middleware/jsx-renderer/index.ts:114-127]

Timeout races downstream execution against a timer and rejects with the configured exception when the timer wins. [src/middleware/timeout/index.ts:38-57] Logger emits an incoming line before calling `next` and an outgoing line containing response status and elapsed time afterward. [src/middleware/logger/index.ts:81-94] Trailing-slash middleware redirects GET and HEAD requests to the selected canonical path either before routing when configured to do so or only after a not-found result. [src/middleware/trailing-slash/index.ts:44-74; src/middleware/trailing-slash/index.ts:128-158]

Configuration

This part reads factory options and HTTP inputs rather than process environment variables. Cache reads the Cache Storage and Web Crypto globals; without Cache Storage it logs or reports the reason and returns a pass-through handler, while missing Web Crypto disables QUERY caching but does not stop ordinary cache operation. [src/middleware/cache/index.ts:183-228]

Language settings include detector order, query and cookie names, header name, path index, supported and fallback languages, case handling, optional conversion, cookie options, and debug output. Validation rejects a fallback absent from the supported set, a negative path index, or an unknown detector type. [src/middleware/language/language.ts:13-68; src/middleware/language/language.ts:204-216]

Security-header configuration merges caller options with defaults and supports content-security policy directives, report endpoints, transport and cross-origin headers, permissions policy, and removal of `X-Powered-By`. The exported `NONCE` handler creates a cryptographically random value when the context does not already hold one, stores it as `secureHeadersNonce`, and formats it for policy directives. [src/middleware/secure-headers/secure-headers.ts:69-145; src/middleware/secure-headers/secure-headers.ts:179-234]

Wiring

The boundary of this part is the Hono context and middleware-handler abstractions imported from the surrounding framework. Middleware depends on context request and response mutation, framework exceptions, router and composition primitives,

helpers for cookies and route matching, and lower-level utilities for cryptography, compression, MIME lookup, body parsing, IP parsing, and URL decoding. [src/middleware/basic-auth/index.ts:6-10; src/middleware/cache/index.ts:6-10; src/middleware/method-not-allowed/index.ts:6-11; src/middleware/serve-static/index.ts:6-11]

Several modules also require platform capabilities at runtime. Context storage imports Node's asynchronous-local-storage API; compression constructs a platform `CompressionStream`; secure headers uses `crypto.getRandomValues`; ETag digesting relies on Web Crypto when no custom digest generator is passed; and static serving delegates actual file or asset retrieval to adapter-supplied `getContent`. [src/middleware/context-storage/index.ts:6-10; src/middleware/compress/index.ts:124-128; src/middleware/secure-headers/secure-headers.ts:131-145; src/middleware/etag/index.ts:38-54; src/middleware/serve-static/index.ts:32-48]

Applications depend on this directory by registering returned handlers around routes and by reading values placed on the context, such as `jwtPayload`, `language`, `requestId`, timing metrics, and the secure-header nonce. [src/middleware/jwt/jwt.ts:145-161; src/middleware/language/language.ts:304-307; src/middleware/request-id/request-id.ts:46-58; src/middleware/timing/timing.ts:85-124; src/middleware/secure-headers/secure-headers.ts:11-15; src/middleware/secure-headers/secure-headers.ts:137-145]

110 entities in `src/middleware`. **1 other subsystem depends on it**, which makes it the 4th most depended-upon part of this codebase.

What it is made of

Its 110 entities sit in 29 files under `src/middleware`: 45 functions, 40 HTTP endpoints, 14 type aliases, 8 constants and 3 more.
`language.ts` holds 16 of them — more than any other file here.

Where work enters

It publishes 40 HTTP endpoints — 24 `GET`, 13 `DELETE`, 2 `PUT` and 1 `ALL`. 2 of them declare a guard — `key` on 2 — and 38 declare none.

- `handler` — `src/middleware/basic-auth/index.ts`:118
- `handler` — `src/middleware/body-limit/index.ts`:72
- `handler` — `src/middleware/cache/index.ts`:79
- `header` — `src/middleware/cache/index.ts`:115

- `handler` — `src/middleware/cache/index.ts`:233
- `directive` — `src/middleware/cache/index.ts`:292

Boundaries

1 other subsystem depends on this one — `Adapter`. Changing what it exposes changes them.

They hold 3 edges into it between them. 33 edges leave it against 3 arriving — it reads more of this repository than this repository reads of it. What they reach is narrower than the folder: 1 of its 110 members carries every inbound edge — `serveStatic` (3). Of the 33 it sends out, 22 go to `Utils` — more than to any other.

It depends on `Utils`, `Helper`, `Jsx`, and on nothing else in this repository.