

---

## A simple API response caching middleware for Express/Node using plain-english durations.

Supports Redis or built-in memory engine with auto-clearing.

downloads 2.2M

coverage 98%

downloads 2.2M

downloads

2.2M

downloads

2.2M

### Why?

Because route-caching of simple data/responses should ALSO be simple.

### Usage

To use, simply inject the middleware (example: `apicache.middleware('5 minutes', [optionalMiddlewareToggle])`) into your routes. Everything else is automagic.

#### Cache a route

```
1 import express from 'express'
2 import apicache from 'apicache'
3
4 let app = express()
5 let cache = apicache.middleware
6
7 app.get('/api/collection/:id?', cache('5 minutes'), (req, res) => {
8   // do some work... this will only occur once per 5 minutes
9   res.json({ foo: 'bar' })
10 })
```

#### Cache all routes

```
1 let cache = apicache.middleware
2
3 app.use(cache('5 minutes'))
4
5 app.get('/will-be-cached', (req, res) => {
6   res.json({ success: true })
7 })
```

#### Use with Redis

```
1 import express from 'express'
2 import apicache from 'apicache'
3 import redis from 'redis'
4
5 let app = express()
6
```

---

```

7 // if redisClient option is defined, apicache will use redis client
8 // instead of built-in memory store
9 let cacheWithRedis = apicache.options({ redisClient: redis.createClient
    () }).middleware
10
11 app.get('/will-be-cached', cacheWithRedis('5 minutes'), (req, res) => {
12   res.json({ success: true })
13 })

```

### Cache grouping and manual controls

```

1 import apicache from 'apicache'
2 let cache = apicache.middleware
3
4 app.use(cache('5 minutes'))
5
6 // routes are automatically added to index, but may be further added
7 // to groups for quick deleting of collections
8 app.get('/api/:collection/:item?', (req, res) => {
9   req.apicacheGroup = req.params.collection
10  res.json({ success: true })
11 })
12
13 // add route to display cache performance (courtesy of @killdash9)
14 app.get('/api/cache/performance', (req, res) => {
15   res.json(apicache.getPerformance())
16 })
17
18 // add route to display cache index
19 app.get('/api/cache/index', (req, res) => {
20   res.json(apicache.getIndex())
21 })
22
23 // add route to manually clear target/group
24 app.get('/api/cache/clear/:target?', (req, res) => {
25   res.json(apicache.clear(req.params.target))
26 })
27
28 /*
29
30 GET /api/foo/bar --> caches entry at /api/foo/bar and adds a group
    called 'foo' to index
31 GET /api/cache/index --> displays index
32 GET /api/cache/clear/foo --> clears all cached entries for 'foo' group/
    collection
33
34 */

```

### Use with middleware toggle for fine control

---

```
1 // higher-order function returns false for responses of other status
  codes (e.g. 403, 404, 500, etc)
2 const onlyStatus200 = (req, res) => res.statusCode === 200
3
4 const cacheSuccesses = cache('5 minutes', onlyStatus200)
5
6 app.get('/api/missing', cacheSuccesses, (req, res) => {
7   res.status(404).json({ results: 'will not be cached' })
8 })
9
10 app.get('/api/found', cacheSuccesses, (req, res) => {
11   res.json({ results: 'will be cached' })
12 })
```

### Prevent cache-control header “max-age” from automatically being set to expiration age

```
1 let cache = apicache.options({
2   headers: {
3     'cache-control': 'no-cache',
4   },
5 }).middleware
6
7 let cache5min = cache('5 minute') // continue to use normally
```

## API

- `apicache.options([globalOptions])` - getter/setter for global options. If used as a setter, this function is chainable, allowing you to do things such as... say... return the middleware.
- `apicache.middleware([duration], [toggleMiddleware], [localOptions])` - the actual middleware that will be used in your routes. `duration` is in the following format “[length][unit]”, as in “10 minutes” or “1 day”. A second param is a middleware toggle function, accepting request and response params, and must return truthy to enable cache for the request. Third param is the options that will override global ones and affect this middleware only.
- `middleware.options([localOptions])` - getter/setter for middleware-specific options that will override global ones.
- `apicache.getPerformance()` - returns current cache performance (cache hit rate)
- `apicache.getIndex()` - returns current cache index [of keys]
- `apicache.clear([target])` - clears cache target (key or group), or entire cache if no value passed, returns new index.
- `apicache.newInstance([options])` - used to create a new ApiCache instance (by default, simply requiring this library shares a common instance)

- `apicache.clone()` - used to create a new ApiCache instance with the same options as the current one

#### Available Options (first value is default)

```
1 {
2   debug:           false|true,      // if true, enables console output
3   defaultDuration: '1 hour',        // should be either a number (in ms
4   enabled:         true|false,      // if false, turns off caching
5   redisClient:     client,          // if provided, uses the [node-
6   appendKey:       fn(req, res),    // appendKey takes the req/res
7   headerBlacklist: [],              // list of headers that should
8   statusCodes: {
9     exclude:        [],              // list status codes to
10    specifically exclude (e.g. [404, 403] cache all responses unless
11    they had a 404 or 403 status)
12    include:         [],              // list status codes to require (e.
13    g. [200] caches ONLY responses with a success/200 code)
14  },
15  trackPerformance: false,           // enable/disable performance
16  headers: {
17    // 'cache-control': 'no-cache' // example of header overwrite
18  },
19  respectCacheControl: false|true    // If true, 'Cache-Control: no-
20  cache' in the request header will bypass the cache.
21 }
```

#### \*Optional: Typescript Types (courtesy of @danielsegl!)

```
1 $ npm install -D @types/apicache
```

## Custom Cache Keys

Sometimes you need custom keys (e.g. save routes per-session, or per method). We've made it easy!

**Note:** All req/res attributes used in the generation of the key must have been set previously (upstream). The entire route logic block is skipped on future cache hits so it can't rely on those params.

```
1 apicache.options({
2   appendKey: (req, res) => req.method + res.session.id,
```

---

```
3  })
```

## Cache Key Groups

Oftentimes it benefits us to group cache entries, for example, by collection (in an API). This would enable us to clear all cached “post” requests if we updated something in the “post” collection for instance. Adding a simple `req.apicacheGroup = [somevalue]`; to your route enables this. See example below:

```
1  var apicache = require('apicache')
2  var cache = apicache.middleware
3
4  // GET collection/id
5  app.get('/api/:collection/:id?', cache('1 hour'), function(req, res,
6    next) {
7    req.apicacheGroup = req.params.collection
8    // do some work
9    res.send({ foo: 'bar' })
10  })
11
12 // POST collection/id
13 app.post('/api/:collection/:id?', function(req, res, next) {
14   // update model
15   apicache.clear(req.params.collection)
16   res.send('added a new item, so the cache has been cleared')
17 })
```

Additionally, you could add manual cache control to the previous project with routes such as these:

```
1  // GET apicache index (for the curious)
2  app.get('/api/cache/index', function(req, res, next) {
3    res.send(apicache.getIndex())
4  })
5
6  // GET apicache index (for the curious)
7  app.get('/api/cache/clear/:key?', function(req, res, next) {
8    res.send(200, apicache.clear(req.params.key || req.query.key))
9  })
```

## Debugging/Console Out

**Using Node environment variables (plays nicely with the hugely popular debug module)**

```
1  $ export DEBUG=apicache
2  $ export DEBUG=apicache,othermoduleThatDebugModuleWillPickUp,etc
```

---

#### By setting internal option

```
1 import apicache from 'apicache'
2
3 apicache.options({ debug: true })
```

### Client-Side Bypass

When sharing [GET](#) routes between admin and public sites, you'll likely want the routes to be cached from your public client, but NOT cached when from the admin client. This is achieved by sending a `"x-apicache-bypass": true` header along with the request from the admin. The presence of this header flag will bypass the cache, ensuring you aren't looking at stale data.

### Contributors

Special thanks to all those that use this library and report issues, but especially to the following active users that have helped add to the core functionality!

- @Chocobozzz - the savior of getting this to pass all the Node 14/15 tests again... thanks for everyone's patience!!!
- @killdash9 - restify support, performance/stats system, and too much else at this point to list
- @svozza - added restify tests, test suite refactor, and fixed header issue with restify. Node v7 + Restify v5 conflict resolution, etag/if-none-match support, etcetc, etc. Triple thanks!!!
- @andredigenova - Added header blacklist as options, correction to caching checks
- @peteboere - Node v7 headers update
- @rutgernation - JSONP support
- @enricsangra - added x-apicache-force-fetch header
- @tskillian - custom appendKey path support
- @agolden - Content-Encoding preservation (for gzip, etc)
- @davidyang - express 4+ compatibility
- @nmors - redis support
- @maytis, @ashwinnaidu - redis expiration
- @ubergesundheit - Corrected buffer accumulation using res.write with Buffers
- @danielsogl - Keeping dev deps up to date, Typescript Types
- @vectart - Added middleware local options support
- @davebaol - Added string support to defaultDuration option (previously just numeric ms)
- @Rauttis - Added ioredis support
- @fernandolguevara - Added opt-out for performance tracking, great emergency fix, thank you!!

---

## Bugfixes, tweaks, documentation, etc.

- @Amhri, @Webcascade, @conmarap, @cjfurelid, @scambier, @lukechilds, @Red-Lv, @gesposito, @viebel, @RowanMeara, @GoingFast, @luin, @keithws, @daveross, @apascal, @guybrush

## Changelog

- **v1.6.0** - added respectCacheControl option flag to force honoring no-cache (thanks @NaridaL!)
- **v1.5.4** - up to Node v15 support, HUGE thanks to @Chocobozzz and all the folks on the PR thread! <3
- **v1.5.3** - multiple fixes: Redis should be connected before using (thanks @guybrush)
- **v1.5.2** - multiple fixes: Buffer deprecation and \_headers deprecation, { trackPerformance: false } by default per discussion (sorry semver...)
- **v1.5.1** - adds { trackPerformance } option to enable/disable performance tracking (thanks @fernandolguevara)
- **v1.5.0** - exposes apicache.getPerformance() for per-route cache metrics (@killdash9 continues to deliver)
- **v1.4.0** - cache-control header now auto-decrements in cached responses (thanks again, @killdash9)
- **v1.3.0** - [securityfix] apicache headers no longer embedded in cached responses when NODE\_ENV === 'production' (thanks for feedback @satya-jugran, @smddzcy, @adamelliot-fields). Updated deps, now requiring Node v6.00+.
- **v1.2.6** - middlewareToggle() now prevents response block on cache hit + falsy toggle (thanks @apascal)
- **v1.2.5** - uses native Node setHeader() rather than express.js header() (thanks @keithws and @daveross)
- **v1.2.4** - force content type to Buffer, using old and new Buffer creation syntax
- **v1.2.3** - add etag to if-none-match 304 support (thanks for the test/issue @svozza)
- **v1.2.2** - bugfix: ioredis.expire params (thanks @GoingFast and @luin)
- **v1.2.1** - Updated deps
- **v1.2.0** - Supports ioredis (thanks @Rauttis)
- **v1.1.1** - bugfixes in expiration timeout clearing and content header preservation under compression (thanks @RowanMeara and @samimakicc).
- **v1.1.0** - added the much-requested feature of a custom appendKey function (previously only took a path to a single request attribute). Now takes (request, response) objects and returns some value to be appended to the cache key.
- **v1.0.0** - stamping v0.11.2 into official production version, will now begin developing on branch v2.x (redesign)

- 
- **v0.11.2** - dev-deps update, courtesy of @danielsogl
  - **v0.11.1** - correction to status code caching, and max-age headers are no longer sent when not cached. middlewareToggle now works as intended with example of statusCode checking (checks during shouldCacheResponse cycle)
  - **v0.11.0** - Added string support to defaultDuration option, previously just numeric ms - thanks @davebaol
  - **v0.10.0** - added ability to blacklist headers (prevents caching) via options.headersBlacklist (thanks @andredigenova)
  - **v0.9.1** - added eslint in prep for v1.x branch, minor ES6 to ES5 in master branch tests
  - **v0.9.0** - corrected Node v7.7 & v8 conflicts with restify (huge thanks to @svozza for chasing this down and fixing upstream in restify itself). Added coveralls. Added middleware.localOptions support (thanks @vectart). Added ability to overwrite/embed headers (e.g. "cache-control": "no-cache") through options.
  - **v0.8.8** - corrected to use node v7+ headers (thanks @peteboere)
  - **v0.8.6, v0.8.7** - README update
  - **v0.8.5** - dev dependencies update (thanks @danielsogl)
  - **v0.8.4** - corrected buffer accumulation, with test support (thanks @ubergesundheit)
  - **v0.8.3** - added tests for x-apicache-bypass and x-apicache-force-fetch (legacy) and fixed a bug in the latter (thanks @Red-Lv)
  - **v0.8.2** - test suite and mock API refactor (thanks @svozza)
  - **v0.8.1** - fixed restify support and added appropriate tests (thanks @svozza)
  - **v0.8.0** - modifies response accumulation (thanks @killdash9) to support res.write + res.end accumulation, allowing integration with restify. Adds gzip support (Node v4.3.2+ now required) and tests.
  - **v0.7.0** - internally sets cache-control/max-age headers of response object
  - **v0.6.0** - removed final dependency (debug) and updated README
  - **v0.5.0** - updated internals to use res.end instead of res.send/res.json/res.jsonp, allowing for any response type, adds redis tests
  - **v0.4.0** - dropped lodash and memory-cache external dependencies, and bumped node version requirements to 4.0.0+ to allow Object.assign native support