

Package: shinyhealth (via r-universe)

October 7, 2026

Title Health Check Endpoints for 'Shiny' Applications

Version 0.0.0.9000

Description Streamlines the ability to add basic health checking for shiny apps.

License MIT + file LICENSE

URL <https://github.com/jimbrig/shinyhealth>,
<http://docs.jimbrig.com/shinyhealth/>

BugReports <https://github.com/jimbrig/shinyhealth/issues>

Depends R (>= 4.2.0)

Imports cli, crayon, rlang (>= 1.1.0), shiny (>= 1.14.0), utils,
yyjsonr

Suggests callr, httpuv, httr2, knitr, later, promises, rmarkdown,
testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/rcmdcheck/ignore-inconsequential-notes true

Config/roxygen2/markdown TRUE

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

Config/pak/sysreqs cmake make libuv1-dev zlib1g-dev

Repository <https://jimbrig.r-universe.dev>

Date/Publication 2026-09-07 03:53:39 UTC

RemoteUrl <https://github.com/jimbrig/shinyhealth>

RemoteRef HEAD

RemoteSha 0adf5a25136f2b1fdf7ae82625c82fe066ca3c85

Contents

add_healthcheck	2
check_inherits	3
has_healthcheck	4
health_handler_default	5
health_response	6
ts_utc	9
Index	10

add_healthcheck	<i>Add a Health Check to a Shiny App</i>
-----------------	--

Description

Registers one or more healthcheck endpoints on a Shiny app by decorating the app object’s internal HTTP handler. Requests matching a registered path (via GET) are answered by handler before any UI rendering, session creation, or authentication occurs; all other requests fall through to the app unchanged.

Because the interception happens on the app object’s own handler, no uiPattern adjustment is required and the endpoint works regardless of how the app was constructed (plain `shiny::shinyApp()`, `golem`, UI wrappers, etc.).

Registrations stack: call `add_healthcheck()` multiple times with different paths to expose e.g. separate **liveness** and **readiness** endpoints (`/livez`, `/readyz`) with their own handlers.

Usage

```
add_healthcheck(  
  app,  
  path = "/health",  
  handler = health_handler_default,  
  verbose = shiny::in_devmode()  
)
```

Arguments

app	A shiny application object, typically created using <code>shiny::shinyApp()</code> .
path	Character vector of URL paths to register (each must begin with <code>/</code>). Default <code>"/health"</code> .
handler	A healthcheck handler: a function of <code>req</code> returning a <code>health_response()</code> , a <code>shiny::httpResponse()</code> , or a named list. See <code>health_handler_default()</code> for the full contract. Errors thrown by the handler are converted into a 503 "fail" response.
verbose	Logical; log a console message for each healthcheck request received. Evaluated once at registration time. Defaults to <code>shiny::in_devmode()</code> .

Value

The modified app object, classed `c("shinyhealth_app", "shiny.appobj")`, with registration metadata stored in the `shinyhealth` attribute.

See Also

[health_response\(\)](#), [health_handler_default\(\)](#), [has_healthcheck\(\)](#), [healthcheck_paths\(\)](#)

Examples

```
app <- shiny::shinyApp(
  ui = shiny::fluidPage("hello"),
  server = function(input, output, session) {}
)

app <- add_healthcheck(app)
has_healthcheck(app)
healthcheck_paths(app)

# stack a second endpoint with a custom handler
app <- add_healthcheck(app, path = "/readyz", handler = function(req) {
  health_response(checks = list(database = list(status = "pass")))
})
healthcheck_paths(app)
```

check_inherits

Class Inheritance Checks

Description

These functions perform checks that assert the underlying class of objects passed to them.

- `check_inherits()`: checks that object `x` is of class `class` using [base::inherits\(\)](#)
- `check_inherits2()`: checks that object `x` is of class `class` using [base::.class2\(\)](#)
- `check_inherits_any()`: checks that object `x` is at least one of the provided classes via [rlang::inherits_any\(\)](#)
- `check_inherits_all()`: checks that object `x` is all of the provided classes via [rlang::inherits_all\(\)](#)

If validation fails for any of these functions, an error is thrown via `check_abort()` displaying a friendly error message.

Usage

```
check_inherits(
  x,
  class,
  arg = rlang::caller_arg(x),
  call = rlang::caller_env()
```

```

)

check_inherits2(
  x,
  class,
  arg = rlang::caller_arg(x),
  call = rlang::caller_env()
)

check_inherits_any(
  x,
  classes,
  arg = rlang::caller_arg(x),
  call = rlang::caller_env()
)

check_inherits_all(
  x,
  classes,
  arg = rlang::caller_arg(x),
  call = rlang::caller_env()
)

```

Arguments

<code>x</code>	The object to check.
<code>class, classes</code>	The name of the class or classes to use during checking.
<code>arg</code>	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
<code>call</code>	The execution environment of a currently running function, e.g. <code>caller_env()</code> . The function will be mentioned in error messages as the source of the error. See the <code>call</code> argument of abort() for more information.

Value

If checks pass, invisibly returns the provided `x` object. If checks fail, a condition error is thrown.

has_healthcheck	<i>Healthcheck Introspection</i>
-----------------	----------------------------------

Description

- `has_healthcheck()`: does the app have at least one registered healthcheck endpoint?
- `healthcheck_paths()`: which paths are registered?

Usage

```
has_healthcheck(app)

healthcheck_paths(app)
```

Arguments

`app` A shiny application object, typically created using [shiny::shinyApp\(\)](#).

Value

- `has_healthcheck()`: a logical scalar.
- `healthcheck_paths()`: a character vector of registered paths (character(0) if none).

Examples

```
app <- shiny::shinyApp(shiny::fluidPage(), function(input, output, session) {})
has_healthcheck(app)

app <- add_healthcheck(app)
has_healthcheck(app)
healthcheck_paths(app)
```

health_handler_default

Default Healthcheck Handler

Description

The default handler used by [add_healthcheck\(\)](#). Returns a healthy [health_response\(\)](#) with no additional fields or checks.

This function also documents the *handler contract*: a healthcheck handler is any function that accepts the incoming request object (`req`) and returns one of:

- a `health_response` (see [health_response\(\)](#))
- a plain [shiny::httpResponse\(\)](#) (passed through untouched)
- a named list, treated as the complete response body. If it contains a `status_code` field, that value is used as the HTTP status code (and removed from the body); a `timestamp` field is added when absent.

Handler return values are normalized via [as_health_response\(\)](#). If a handler throws an error, [add_healthcheck\(\)](#) converts it into a 503 response with `status = "fail"` and the condition message under output – the endpoint stays machine-readable even when a check is broken.

Usage

```
health_handler_default(req)
```

Arguments

req A shiny request object

Value

A health_response object.

See Also

[add_healthcheck\(\)](#), [health_response\(\)](#)

Examples

```
health_handler_default(req = NULL)
```

health_response	<i>Health Check Response</i>
-----------------	------------------------------

Description

Functions for creating and working with health_response objects, the responses served from registered healthcheck endpoints:

- `health_response()`: user-facing helper that assembles a response body, applies defaults, validates inputs, and constructs the object.
- `new_health_response()`: low-level constructor for developers extending the class. Performs no validation or defaulting beyond serialization.
- `validate_health_response()`: structural validator.
- `is_health_response()`: predicate.
- `as_health_response()`: coercion generic used to normalize healthcheck handler return values.

A health_response is a [shiny::httpResponse\(\)](#) (classed c("health_response", "httpResponse")), so shiny's internal HTTP machinery serves it like any other response. The structured (pre-serialization) response body is retained in the body attribute for printing, inspection, and testing.

Usage

```
health_response(
  status = "pass",
  ...,
  checks = NULL,
  status_code = 200L,
  timestamp = ts_utc(),
  headers = list(),
  content_type = "application/health+json"
)
```

```

new_health_response(
  body = list(),
  status_code = 200L,
  content_type = "application/health+json",
  headers = list()
)

validate_health_response(
  x,
  arg = rlang::caller_arg(x),
  call = rlang::caller_env()
)

is_health_response(x)

as_health_response(x, ...)

```

Arguments

status	Character string reported as the body status field. Default "pass".
...	Additional named fields merged into the response body at the top level (e.g. version, description, output).
checks	Optional named list of check results, serialized under the body's checks field. Elements may be arbitrarily nested lists; their structure is not interpreted.
status_code	Integer HTTP status code between 100 and 599. Default 200L.
timestamp	Character timestamp for the body's timestamp field. Default <code>ts_utc()</code> .
headers	Named list of HTTP headers merged over the defaults (caller wins).
content_type	MIME type for the response. Default "application/health+json".
body	Named list representing the structured response body (low-level constructor).
x	An object to coerce or test.
arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
call	The execution environment of a currently running function, e.g. <code>caller_env()</code> . The function will be mentioned in error messages as the source of the error. See the <code>call</code> argument of <code>abort()</code> for more information.

Details

Status Semantics:

The HTTP transport status (`status_code`) and the body status field are deliberately independent:

- Orchestrators and load balancers ([Kubernetes](#), [Cloud Run](#), [Azure Container Apps](#), etc.) interpret only the *HTTP status code*: codes ≥ 200 and < 400 indicate success, anything else indicates failure.

- The body status string is a convention for humans and tooling. The default value ("pass") follows the [IETF health check response format draft](#) (draft-inadarei-api-health-check, which also suggests "warn" and "fail"), but any value is accepted – use whatever vocabulary fits your conventions.

The defaults compose into a healthy response (status = "pass", status_code = 200L). For an unhealthy response, 503L (Service Unavailable) is the standard choice, typically paired with status = "fail". How individual checks aggregate into an overall status is intentionally *not* decided here – that policy belongs to the handler that builds the response.

Headers, Content Type, & Timestamps:

- Default headers are Cache-Control: no-store ([RFC 9111](#)) so probe responses are never cached. The legacy Pragma/Expires response headers are deprecated by RFC 9111 and intentionally omitted. CORS headers are not sent by default (probes are server-to-server); supply them via headers if a browser client on another origin needs access.
- The default content type is application/health+json per the IETF draft. The +json structured syntax suffix ([RFC 6839](#)) guarantees any JSON processor handles it; use content_type to override (e.g. plain application/json).
- Timestamps use [RFC 3339](#) / ISO 8601 format in UTC (see [ts_utc\(\)](#)).

Value

- health_response(), new_health_response(), and as_health_response() return a health_response object (a classed [shiny::httpResponse\(\)](#)).
- validate_health_response() invisibly returns its input if valid, otherwise throws a check_error condition.
- is_health_response() returns a logical scalar.

Examples

```
# healthy response with defaults
health_response()

# unhealthy response (standard choice: 503 + "fail")
health_response(status = "fail", status_code = 503L)

# custom vocabulary, extra fields, and nested checks
health_response(
  status = "HEALTHY",
  version = "1.0.0",
  checks = list(
    database = list(status = "pass", latency_ms = 2.3),
    api = list(status = "warn", output = "slow response times")
  )
)
```

`ts_utc`*UTC Timestamp*

Description

Formats a time as an **RFC 3339** / ISO 8601 timestamp in UTC (e.g. "2026-09-06T21:15:30Z"), the interchange format expected in machine-readable API responses.

Named `ts_utc()` (rather than `ts()`) to avoid masking `stats::ts()`.

Usage

```
ts_utc(time = Sys.time())
```

Arguments

`time` A **POSIXct** time. Defaults to `Sys.time()`.

Value

A character string containing the formatted timestamp.

Examples

```
ts_utc()
```

Index

`abort()`, [4](#), [7](#)
`add_healthcheck`, [2](#)
`add_healthcheck()`, [5](#), [6](#)
`as_health_response (health_response)`, [6](#)
`as_health_response()`, [5](#)

`base:::class2()`, [3](#)
`base:::inherits()`, [3](#)

`check_inherits`, [3](#)
`check_inherits2 (check_inherits)`, [3](#)
`check_inherits_all (check_inherits)`, [3](#)
`check_inherits_any (check_inherits)`, [3](#)

`has_healthcheck`, [4](#)
`has_healthcheck()`, [3](#)
`health_handler_default`, [5](#)
`health_handler_default()`, [2](#), [3](#)
`health_response`, [6](#)
`health_response()`, [2](#), [3](#), [5](#), [6](#)
`healthcheck_paths (has_healthcheck)`, [4](#)
`healthcheck_paths()`, [3](#)

`is_health_response (health_response)`, [6](#)

`new_health_response (health_response)`, [6](#)

`POSIXct`, [9](#)

`rlang::inherits_all()`, [3](#)
`rlang::inherits_any()`, [3](#)

`shiny::httpResponse()`, [2](#), [5](#), [6](#), [8](#)
`shiny::in_devmode()`, [2](#)
`shiny::shinyApp()`, [2](#), [5](#)
`stats::ts()`, [9](#)
`Sys.time()`, [9](#)

`ts_utc`, [9](#)
`ts_utc()`, [7](#), [8](#)

`validate_health_response`
 (`health_response`), [6](#)