

Package ‘beakr’

July 22, 2025

Type Package

Title A Minimalist Web Framework for R

Version 0.4.3

Author Hans Martin [aut],
Jonathan Callahan [aut, cre]

Maintainer Jonathan Callahan <jonathan.s.callahan@gmail.com>

Description A minimalist web framework for developing application programming interfaces in R that provides a flexible framework for handling common HTTP-requests, errors, logging, and an ability to integrate any R code as server middle-ware.

License GPL-3

URL <https://github.com/MazamaScience/beakr>

BugReports <https://github.com/MazamaScience/beakr/issues>

ByteCompile TRUE

Depends R (>= 3.1.0)

Imports R6, base64enc, httpuv, jsonlite, magrittr, mime, stringr,
webutils

Suggests knitr, testthat, rmarkdown

Encoding UTF-8

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2021-04-06 17:20:02 UTC

Contents

beakr-package	2
Beakr	3
cors	4

decorate	6
Error	7
handleErrors	8
httpDELETE	9
httpGET	10
httpPOST	11
httpPUT	12
jsonError	13
listen	14
Listener	15
listServers	16
Middleware	17
newBeakr	18
Request	19
Response	20
Router	23
serveStaticFiles	24
stopAllServers	26
stopServer	26
Index	28

beakr-package	<i>A minimalist web framework.</i>
---------------	------------------------------------

Description

The **beakr** package provides a minimalist web framework for for developing application programming interfaces in R. The package includes basic functionality for handling common HTTP requests.

beakr allows R code to listen for and respond to HTTP requests, so you can serve web traffic directly from an R process. **beakr** relies heavily on the **httpuv** package, and therefore the lower level **libuv** and **http-parser** C libraries. **beakr** is a ground-up rewrite and continuation of the **jug** package developed by Bart Smeets. The **beakr** package is supported and maintained by **Mazama Science**.

Author(s)

Hans Martin <hans@mazamascience.com>

See Also

[newBeakr](#)

Beakr	<i>Beakr Application class</i>
-------	--------------------------------

Description

A Beakr object defines the server instance utilizing the **httpuv** package. This class defines an interface for the rest of the **beakr** package and is therefore meant to be instantiated.

Methods

`router()` An instantiated Router object.
`server()` The instantiated Server object.
`appDefinition()` A method to define the functions or middleware of users application.
`initialize()` Creates a new Router object for the router method.
`start(host, port, daemon)` Returns a running server. If `daemon = TRUE`, the server will run in the background.
`print(...)` Returns a console output of the instance and its number of middleware attached.

Package details

The **beakr** package provides a minimal web framework for for developing lightweight APIs in R. The package includes basic functionality for handling common HTTP requests. **beakr** is a ground-up rewrite and continuation of the **jug** package developed by Bart Smeets. The **beakr** package is supported and maintained by [Mazama Science](#).

Methods

Public methods:

- `Beakr$appDefinition()`
- `Beakr$new()`
- `Beakr$start()`
- `Beakr$print()`
- `Beakr$clone()`

Method `appDefinition()`:

Usage:

`Beakr$appDefinition()`

Method `new()`:

Usage:

`Beakr$new()`

Method `start()`:

Usage:

```
Beakr$start(host, port, daemon)
```

Method `print()`:

Usage:

```
Beakr$print()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Beakr$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

[Router](#) and [Middleware](#)

cors

Allow Cross-Origin-Requests

Description

Allow Cross-Origin Resource Sharing headers as described in [MDN Web Docs](#). Cross-origin resource sharing is a mechanism that allows restricted resources on a web page to be requested from another domain(origin) outside the domain from which the first resource was served.

Usage

```
cors(
  beakr,
  path = NULL,
  methods = c("GET", "POST", "PUT", "DELETE", "OPTIONS", "PATCH"),
  origin = "*",
  credentials = NULL,
  headers = NULL,
  maxAge = NULL,
  expose = NULL
)
```

Arguments

<code>beakr</code>	Beakr instance object.
<code>path</code>	String representing a path for which to specify a CORS policy. Default NULL applies a single policy for all URL routes.
<code>methods</code>	A vector of the request methods to allow. i.e Access-Control-Allow-Methods parameter, e.g GET, POST.

origin	A vector of the request origin(s) for which resource sharing is enabled. i.e Access-Control-Allow-Origin response header parameter.
credentials	A boolean to enable/disable credentialed requests. i.e Access-Control-Allow-Credentials response header parameter.
headers	A vector of the allowed headers. i.e Access-Control-Allow-Headers response header parameter.
maxAge	The max age, in seconds. i.e Access-Control-Max-Age response header parameter.
expose	The headers to expose. i.e Access-Control-Expose-Headers response header parameter.

Value

A Beakr instance with CORS enabled

Note

You can verify that CORS is enabled by using the Chrome browser and opening up the Developer Tools. The "Network" tab allows you to inspect response headers and see where the Cross-Origin policy is specified.

If you run the example in the console, be sure to `stopServer(bekar)` when you are done.

See Also

[Request](#), [Response](#), [Error](#)

Examples

```
library(beakr)

# Create an new beakr instance
beakr <- newBeakr()

# beakr pipeline
beakr %>%

  # Enable CORS
  cors() %>%

  # Respond to GET requests at the "/hi" route
  httpGET(path = "/hi", function(req, res, err) {
    print("Hello, World!")
  }) %>%

  # Respond to GET requests at the "/bye" route
  httpGET(path = "/bye", function(req, res, err) {
    print("Farewell, my friends.")
  }) %>%

  # Start the server on port 25118
```

```
listen(host = "127.0.0.1", port = 25118, daemon = TRUE)

# -----
# POINT YOUR BROWSER AT:
# * http://127.0.0.1:25118/hi
# * http://127.0.0.1:25118/bye
#
# THEN, STOP THE SERVER WITH stopServer(beakr)
# -----

# Stop the beakr instance server
stopServer(beakr)
```

 decorate

Decorate a function for use in a web service

Description

The `decorate()` function can be used to prepare a function for easy use in a beakr pipeline.

Decorating a function associates the specified function and its parameters with `req`, `res`, and `err` objects and assigns a content-type to the response object. This prepares a standard R function to be used in Beakr instances and accept requests.

Usage

```
decorate(FUN, content_type = "text/html", strict = FALSE)
```

Arguments

<code>FUN</code>	Function to decorate.
<code>content_type</code>	HTTP "content-type" of the function output. (e.g. "text/plain", "text/html" or other mime type)
<code>strict</code>	Boolean, requiring strict parameter matching.

Value

A *decorated* middleware function.

Examples

```
library(beakr)

# Create an new Beakr instance
beakr <- newBeakr()

# Create simple hello and goodbye function
hello <- function(name) { paste0("Hello, ", name, "!" ) }
```

```

goodbye <- function(text = "Adios") { paste0(text, ", dear friend.") }

# Create a web service from these functions
beakr %>%

  httpGET(path = "/hello", decorate(hello)) %>%

  httpGET(path = "/goodbye", decorate(goodbye)) %>%

  handleErrors() %>%

  listen(host = '127.0.0.1', port = 25118, daemon = TRUE)

# -----
# POINT YOUR BROWSER AT:
# * http://127.0.0.1:25118/hello?name=Honeydew
# * http://127.0.0.1:25118/goodbye?text=Sionara
#
# THEN, STOP THE SERVER WITH stopServer(beakr)
# -----

# Stop the beakr instance server
stopServer(beakr)

```

Error

Error class

Description

An Error object represents the state and handling of instance or middleware errors.

Fields

`errors` Returns a list of errors, if any.

`occured` Returns TRUE if any error has occurred, FALSE otherwise.

Methods

`set(err)` Sets an error.

Methods

Public methods:

- [Error\\$set\(\)](#)
- [Error\\$clone\(\)](#)

Method `set()`:

Usage:

```
Error$set(err)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Error$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

[handleErrors](#) and [Middleware](#)

handleErrors	<i>Error-handling middleware</i>
--------------	----------------------------------

Description

This default error handler should be added at the end of the beakr pipeline, right before `listen()`. Errors generated by any previous step will be returned within a JSON wrapper.

Usage

```
handleErrors(beakr = NULL, FUN = jsonError)
```

Arguments

<code>beakr</code>	Beakr instance
<code>FUN</code>	a function to handle the error response

Value

A Beakr instance with added middleware.

Note

If you run the example in the console, be sure to `stopServer(bekar)` when you are done.

Examples

```
library(beakr)

# Create an new beakr instance
beakr <- newBeakr()

# beakr pipeline
beakr %>%

# Respond to GET requests at the "/hi" route
```

```

    httpGET(path = "/hi", function(req, res, err) {
        print("Hello, World!")
    }) %>%

    # Respond to GET requests at the "/bye" route
    httpGET(path = "/bye", function(req, res, err) {
        print("Farewell, my friends.")
    }) %>%

    handleErrors() %>%

    # Start the server on port 25118
    listen(host = "127.0.0.1", port = 25118, daemon = TRUE)

    # -----
    # POINT YOUR BROWSER AT:
    # * http://127.0.0.1:25118/NOT_A_ROUTE
    #
    # THEN, STOP THE SERVER WITH stopServer(beakr)
    # -----

    # Stop the beakr instance server
    stopServer(beakr)

```

httpDELETE

DELETE-binding middleware

Description

Routes HTTP DELETE requests to the specified path with the specified callback functions or middleware.

Usage

```
httpDELETE(beakr, path = NULL, FUN = NULL)
```

Arguments

beakr	Beakr instance or NULL.
path	String representing a path for which the middleware function is invoked.
FUN	Middleware function to be invoked.

Value

A Beakr instance with added middleware.

Examples

```
## Not run:
library(beakr)

# Create an new Beakr instance
beakr <- newBeakr()

# Create a simple beakr pipeline
beakr %>%
  httpDELETE("/", function(req, res, err) {
    return("Successful DELETE request!\n")
  }) %>%
  listen(host = '127.0.0.1', port = 25118, daemon = TRUE)

# -----
# IN A TERMINAL:
# curl -X DELETE http://127.0.0.1:25118/
# > Successful DELETE request!
# -----

# Stop the beakr instance server
stopServer(beakr)

## End(Not run)
```

httpGET

GET-binding middleware

Description

Routes HTTP GET requests to the specified path with the specified callback functions or middleware.

Usage

```
httpGET(beakr, path = NULL, FUN = NULL)
```

Arguments

beakr	Beakr instance or NULL.
path	String representing a path for which the middleware function is invoked.
FUN	Middleware function to be invoked.

Value

A Beakr instance with added middleware.

Examples

```
## Not run:
library(beakr)

# Create an new Beakr instance
beakr <- newBeakr()

# Create a simple beakr pipeline
beakr %>%
  httpGET("/", function(req, res, err) {
    return("Successful GET request!\n")
  }) %>%
  listen(host = '127.0.0.1', port = 25118, daemon = TRUE)

# -----
# IN A TERMINAL:
# curl -X GET http://127.0.0.1:25118/
# > Successful GET request!
# -----

# Stop the beakr instance server
stopServer(beakr)

## End(Not run)
```

httpPOST

POST-binding middleware

Description

Routes HTTP POST requests to the specified path with the specified callback functions or middle-ware.

Usage

```
httpPOST(beakr, path = NULL, FUN = NULL)
```

Arguments

beakr	Beakr instance or NULL.
path	String representing a path for which the middleware function is invoked.
FUN	Middleware function to be invoked.

Value

A Beakr instance with added middleware.

Examples

```
## Not run:
library(beakr)

# Create an new Beakr instance
beakr <- newBeakr()

# Create a simple beakr pipeline
beakr %>%
  httpPOST("/", function(req, res, err) {
    return("Successful POST request!\n")
  }) %>%
  listen(host = '127.0.0.1', port = 25118, daemon = TRUE)

# -----
# IN A TERMINAL:
# curl -X POST http://127.0.0.1:25118/
# > Successful POST request!
# -----

# Stop the beakr instance server
stopServer(beakr)

## End(Not run)
```

httpPUT

PUT-binding middleware

Description

Routes HTTP PUT requests to the specified path with the specified callback functions or middleware.

Usage

```
httpPUT(beakr, path = NULL, FUN = NULL)
```

Arguments

beakr	Beakr instance or NULL.
path	String representing a path for which the middleware function is invoked.
FUN	Middleware function to be invoked.

Value

A Beakr instance with added middleware.

Examples

```
## Not run:
library(beakr)

# Create an new Beakr instance
beakr <- newBeakr()

# Create a simple beakr pipeline
beakr %>%
  httpPUT("/", function(req, res, err) {
    return("Successful PUT request!\n")
  }) %>%
  listen(host = '127.0.0.1', port = 25118, daemon = TRUE)

# -----
# IN A TERMINAL:
# curl -X PUT http://127.0.0.1:25118/
# > Successful PUT request!
# -----

# Stop the beakr instance server
stopServer(beakr)

## End(Not run)
```

jsonError*JSON error function*

Description

This function is used to add a JSON error response to the res object. It is called by the `handleErrors()` utility function.

Usage

```
jsonError(req, res, err)
```

Arguments

req	Request object.
res	Response object.
err	Error Error object.

Value

The incoming res object is modified.

See Also

[Request](#), [Response](#), [Error](#)

listen	<i>Listen for connections on a Beakr instance</i>
--------	---

Description

Binds and listens for connections at the specified host and port.

Usage

```
listen(
  beakr = NULL,
  host = "127.0.0.1",
  port = 25118,
  daemon = FALSE,
  verbose = FALSE
)
```

Arguments

beakr	Beakr instance.
host	String that is a valid IPv4 or IPv6 address to listen on. Defaults to the local host ("127.0.0.1").
port	Number or integer that indicates the port to listen on. Default is a port opened on 25118.
daemon	Logical specifying whether the server should be run in the background.
verbose	Logical specifying whether to print out details of the Beakr instance now running. This should only be used when running a beaker app interactively, not in production.

Details

listen() binds the specified host and port and listens for connections on a thread. The thread handles incoming requests. when it receives an HTTP request, it will schedule a call to the user-defined middleware and handle the request.

If daemon = TRUE, listen() binds the specified port and listens for connections on a thread running in the background.

See the **httpuv** package for more details.

Value

A Beakr instance with an active server.

Note

The default port number 25118 was generated using:

```
> match(c("b","e","a","k","r"), letters) %% 10
[1] 2 5 1 1 8
```

Examples

```
library(beakr)

# Create an new Beakr instance
beakr <- newBeakr()

# beakr pipeline
beakr %>%

  httpGET("/", function(req, res, err) {
    return("Successful GET request!\n")
  }) %>%

  listen(daemon = TRUE)      # run in the background

# Stop the server
stopServer(beakr)
```

Listener

Listener class

Description

A Listener object provides a simple, programmatically controlled HTTP protocol listener.

Fields

`FUN` Returns function response.

`event` Returns event type.

Methods

`initialize(FUN, event)` Sets instance object function and event state.

Methods**Public methods:**

- `Listener$new()`
- `Listener$clone()`

Method `new()`:

Usage:

```
Listener$new(event, FUN, ...)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Listener$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

[Router](#) and [Error](#)

listServers

List all servers

Description

Lists all Beakr servers currently running (and any other servers created with the **httpuv** package). This function is included to encourage experimentation so that users who create multiple Beakr instances can quickly find and stop them all.

See `httpuv::listServers` for details.

Usage

```
listServers()
```

Value

None

Examples

```
library(beakr)

beakr1 <- newBeakr()
beakr2 <- newBeakr()
beakr1 %>% listen(daemon = TRUE, port = 1234, verbose = TRUE)
beakr2 %>% listen(daemon = TRUE, port = 4321, verbose = TRUE)
length(listServers())
stopAllServers()
length(listServers())
```

Middleware

Middleware class

Description

A Middleware object represents middleware functions that have access to the request (req), response (res) and error (err) objects in request-response cycle via the Router.

Methods

`path` Returns the path for the specified middleware.

`FUN` Returns the function response.

`method` Returns the HTTP method for the middleware, i.e. "GET", "POST", etc.

`protocol` Returns the protocol, "http" or "websocket".

`initialize(FUN, path, method, websocket)` Initializes the state of new middleware.

Methods

Public methods:

- [Middleware\\$new\(\)](#)
- [Middleware\\$clone\(\)](#)

Method `new()`:

Usage:

`Middleware$new(FUN, path, method, websocket)`

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`Middleware$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

[Router](#) and [Middleware](#)

newBeakr

*Create a new Beakr instance***Description**

Create a Beakr instance by calling the top-level `newBeakr()` function. If name is not supplied, a random name will be assigned.

This Beakr instance will then begin a pipeline of separate middleware steps for routing, serving files and handling errors. The pipeline will end with the `listen()` function.

Usage

```
newBeakr(name = NULL)
```

Arguments

name Optional name assigned to the Beakr instance.

Value

A new and empty Beakr instance.

Examples

```
library(beakr)

# Create an new beakr instance
beakr <- newBeakr()

# beakr pipeline of handlers
beakr %>%

  httpGET(path = "/route_A", function(res, req, err) {
    print("This is route 'A'.")
  }) %>%

  httpGET(path = "/route_B", function(res, req, err) {
    print("This is route 'B'.")
  }) %>%

  handleErrors() %>%

  listen(host = '127.0.0.1', port = 25118, daemon = TRUE)

# -----
# POINT YOUR BROWSER AT:
# * http://127.0.0.1:25118/route_A
# * http://127.0.0.1:25118/route_B
#
```

```
# THEN, STOP THE SERVER WITH stopServer(beakr)
# -----

# Stop the beakr instance server
stopServer(beakr)
```

Request

*Request Class***Description**

A Request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on. In this documentation and by convention, the object is always referred to as req (and the HTTP response is res).

Fields

parameters A list containing properties mapped to the named router parameters.

headers A list of response headers.

path Contains the path part of the request URL.

method Contains a string corresponding to the HTTP method of the request: GET, POST, PUT, and so on.

raw Returns the raw request (req) object.

type Contains the body content-type, i.e. "text/html" or "application/json".

body Contains the data submitted in the request body.

protocol Contains the request protocol string.

Methods

attach(key, value) Returns a key-value.

getHeader(key) Returns the key element of the headers list.

setHeader(key, value) Attaches a header to headers list.

addParameters(named_list) Adds parameters to the named key-value parameters list.

intialize(req) Creates a new Request object by parsing and extracting features of req input and populating the object fields.

Methods**Public methods:**

- [Request\\$attach\(\)](#)
- [Request\\$getHeader\(\)](#)
- [Request\\$setHeader\(\)](#)
- [Request\\$addParameters\(\)](#)

- [Request\\$new\(\)](#)
- [Request\\$clone\(\)](#)

Method attach():*Usage:*`Request$attach(key, value)`**Method** getHeader():*Usage:*`Request$getHeader(key)`**Method** setHeader():*Usage:*`Request$setHeader(key, value)`**Method** addParameters():*Usage:*`Request$addParameters(named_list)`**Method** new():*Usage:*`Request$new(req)`**Method** clone(): The objects of this class are cloneable with this method.*Usage:*`Request$clone(deep = FALSE)`*Arguments:*`deep` Whether to make a deep clone.**See Also**[Response](#)

`Response`*Response Class*

Description

A `Response` object represents the HTTP response that a Beakr sends when it gets an HTTP request. It is by convention, the object is always referred to as `res` (and the HTTP request is `req`).

Fields

`headers` A list containing a key-value header list.

`status` An integer HTTP status code.

`body` Contains the response body.

Methods

`setHeader(key, value)` Sets a key-value header, i.e. "Content-Type" = "text/html".

`setContentType(type)` Sets the response content-type.

`setStatus(status)` Sets the HTTP status code.

`setBody(body)` Sets the body response.

`redirect(url)` Sets the HTTP status to 302, "Found" and redirects to url.

`json(txt, auto_unbox = TRUE)` Applies a function to text convert to JSON and sets the content-type to JSON.

`text(txt)` Sets the response body text.

`structured(protocol)` Sets the response protocol, i.e. "http"

`plot(plot_object, base64 = TRUE, ...)` Sets the response type to plot image output.

Methods**Public methods:**

- [Response\\$setHeader\(\)](#)
- [Response\\$setContentType\(\)](#)
- [Response\\$setStatus\(\)](#)
- [Response\\$setBody\(\)](#)
- [Response\\$redirect\(\)](#)
- [Response\\$json\(\)](#)
- [Response\\$text\(\)](#)
- [Response\\$structured\(\)](#)
- [Response\\$plot\(\)](#)
- [Response\\$clone\(\)](#)

Method setHeader():

Usage:

`Response$setHeader(key, value)`

Method setContentType():

Usage:

`Response$setContentType(type)`

Method setStatus():

Usage:

`Response$setStatus(status)`

Method `setBody()`:*Usage:*`Response$setBody(body)`**Method** `redirect()`:*Usage:*`Response$redirect(url)`**Method** `json()`:*Usage:*`Response$json(txt, auto_unbox = TRUE)`**Method** `text()`:*Usage:*`Response$text(txt)`**Method** `structured()`:*Usage:*`Response$structured(protocol)`**Method** `plot()`:*Usage:*`Response$plot(plot_object, base64 = TRUE, ...)`**Method** `clone()`: The objects of this class are cloneable with this method.*Usage:*`Response$clone(deep = FALSE)`*Arguments:*`deep` Whether to make a deep clone.**See Also**[Response](#)

Router

*Router Class***Description**

A Router object represents the handling of routing and middleware (such as `httpGET()`, `httpPUT()`, `httpPOST()`, and so on). Once a Router object is instantiated, middleware and HTTP method routes can be added. The top level Beakr object initializes with the creation of a Router object.

Fields

`middleware` A list of specified middleware function or functions.

`listeners` A list of specified listeners.

Methods

`addMiddleware(middleware)` A method to add middleware function(s) to middleware.

`addListener(listener)` A method to add listeners to listeners.

`processEvent(event, ...)` Processes the event heard by the Listener.

`invoke(req, websocket_msg, websocket_binary)` This method is used to create the request-response cycle objects of the provided middleware.

Methods**Public methods:**

- [Router\\$addMiddleware\(\)](#)
- [Router\\$addListener\(\)](#)
- [Router\\$processEvent\(\)](#)
- [Router\\$invoke\(\)](#)
- [Router\\$clone\(\)](#)

Method `addMiddleware()`:

Usage:

`Router$addMiddleware(middleware)`

Method `addListener()`:

Usage:

`Router$addListener(listener)`

Method `processEvent()`:

Usage:

`Router$processEvent(event, ...)`

Method `invoke()`:

Usage:

```
Router$invoke(req, websocket_msg = NULL, websocket_binary = NULL)
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
Router$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

[Response](#)

serveStaticFiles	<i>File-serving middleware</i>
------------------	--------------------------------

Description

Binds to GET requests that aren't handled by specified paths. The result is to return files that are found on the host machine at the requested path. Binary file types like .png, .gif or .pdf are returned as raw bytes. All others are returned as characters.

Mime types are guessed using the **mime** package. The rawTypesPattern parameter is used to match mime types that should be returned as raw bytes.

Usage

```
serveStaticFiles(
  beakr = NULL,
  urlPath = NULL,
  rootPath = getwd(),
  rawTypesPattern = "image|json|octet|pdf|video",
  verbose = FALSE
)
```

Arguments

beakr	Beakr instance or NULL.
urlPath	String representing the URL directory underneath which static file paths will appear.
rootPath	String representing the absolute path used as the root directory when searching for files on host machine. Defaults to the directory in which the script is running.
rawTypesPattern	String pattern identifying mime types to be returned as raw bytes.
verbose	Boolean to show a verbose static file information.

Details

All files to be served in this manner must exist underneath the host machine directory specified with `rootPath`. The directory structure underneath `rootPath` will be mapped onto URLs underneath `urlPath`. This helps when deploying web services at preordained URLs.

The example below presents files underneath host machine directory `hostDir/` to be accessed at URLs under `test/`.

Value

A Beakr instance with added middleware.

Note

If you run the example in the console, be sure to `stopServer(bekar)` when you are done.

Examples

```
library(beakr)

# Create a .txt file in temp directory
hostDir <- tempdir()
file <- paste0(hostDir, "/my_file.txt")
cat("I am a text file.", file = file)

# Create an new beakr instance
beakr <- newBeakr()

# beakr pipeline
beakr %>%

  # Respond to GET requests at the "/hi" route
  httpGET(path = "/hi", function(req, res, err) {
    print("Hello, World!")
  }) %>%

  # Respond to GET requests at the "/bye" route
  httpGET(path = "/bye", function(req, res, err) {
    print("Farewell, my friends.")
  }) %>%

  # Host the directory of static files
  serveStaticFiles("/test", hostDir, verbose = TRUE) %>%

  # Start the server on port 25118
  listen(host = "127.0.0.1", port = 25118, daemon = TRUE)

# -----
# POINT YOUR BROWSER AT:
# * http://127.0.0.1:25118/test/my_file.txt
#
# THEN, STOP THE SERVER WITH stopServer(beakr)
```

```
# -----
# Stop the beakr instance server
stopServer(beakr)
```

stopAllServers	<i>Stop all servers</i>
----------------	-------------------------

Description

Stops all Beakr servers currently running (and any other servers created with the **httpuv** package). This function is included to encourage experimentation so that users who create multiple Beakr instances can quickly find and stop them all.

See `httpuv::stopAllServers` for details.

Usage

```
stopAllServers()
```

Value

None

Examples

```
library(beakr)

beakr1 <- newBeakr()
beakr2 <- newBeakr()
beakr1 %>% listen(daemon = TRUE, port = 1234, verbose = TRUE)
beakr2 %>% listen(daemon = TRUE, port = 4321, verbose = TRUE)
length(listServers())
stopAllServers()
length(listServers())
```

stopServer	<i>Stop a beakr instance server</i>
------------	-------------------------------------

Description

Stops the server associated with a Beakr instance, closing all open connections and unbinding the port.

Usage

```
stopServer(beakr = NULL, verbose = FALSE)
```

Arguments

<code>beakr</code>	Beakr instance.
<code>verbose</code>	Logical specifying whether to print out details of the Beakr instance just stopped.

Value

None

Examples

```
library(beakr)

beakr <- newBeakr()

# beakr pipeline
beakr %>%

  handleErrors() %>%

  listen(daemon = TRUE, verbose = TRUE)

stopServer(beakr, verbose = TRUE)
```

Index

*** package**

beakr-package, [2](#)

Beakr, [3](#)

Beakr-Package (beakr-package), [2](#)

beakr-package, [2](#)

cors, [4](#)

decorate, [6](#)

Error, [5](#), [7](#), [14](#), [16](#)

handleErrors, [8](#), [8](#)

httpDELETE, [9](#)

httpGET, [10](#)

httpPOST, [11](#)

httpPUT, [12](#)

jsonError, [13](#)

listen, [14](#)

Listener, [15](#)

listServers, [16](#), [16](#)

Middleware, [4](#), [8](#), [17](#), [17](#)

newBeakr, [2](#), [18](#)

Request, [5](#), [14](#), [19](#)

Response, [5](#), [14](#), [20](#), [20](#), [22](#), [24](#)

Router, [4](#), [16](#), [17](#), [23](#)

serveStaticFiles, [24](#)

stopAllServers, [26](#), [26](#)

stopServer, [26](#)