Package 'ambiorix'

January 27, 2021

Title Web Framework Inspired by 'Express.js'

Version 1.0.2

Description A web framework inspired by 'express.js' to build any web service from multi-page websites to 'RESTful' application programming interfaces.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Imports fs, log, cli, glue, here, httpuv, methods, promises, jsonlite, websocket, assertthat

Suggests mime, readr, htmltools, htmlwidgets

URL https://github.com/JohnCoene/ambiorix, https://ambiorix.john-coene.com

BugReports https://github.com/JohnCoene/ambiorix/issues

NeedsCompilation no

Author John Coene [aut, cre] (<https://orcid.org/0000-0002-6637-4107>)

Maintainer John Coene <jcoenep@gmail.com>

Repository CRAN

Date/Publication 2021-01-27 10:00:07 UTC

R topics documented:

`Ambiorix` *Ambiorix*

Description

Web server.

Value

An object of class `Ambiorix` from which one can add routes, routers, and run the application.

Public fields

`not_found` 404 Response, must be a handler function that accepts the request and the response, by default uses `response_404()`.

`is_running` Boolean indicating whether the server is running.

`error` 500 response when the route errors, must a handler function that accepts the request and the response, by default uses `response_500()`.

`on_stop` Callback function to run when the app stops, takes no argument.

Active bindings

`websocket` A handler function that accepts a websocket which overrides ambiorix internal websocket handling.

Methods

Public methods:

- `Ambiorix$new()`
- `Ambiorix$listen()`
- `Ambiorix$get()`
- `Ambiorix$put()`
- `Ambiorix$patch()`
- `Ambiorix$delete()`
- `Ambiorix$post()`
- `Ambiorix$all()`
- `Ambiorix$set_404()`

- Ambiorix$static()
- Ambiorix$start()
- Ambiorix$receive()
- Ambiorix$serialiser()
- Ambiorix$stop()
- Ambiorix$print()
- Ambiorix$use()
- Ambiorix$clone()

Method `new()`**:**

Usage:

```
Ambiorix$new(
  host = getOption("ambiorix.host", "0.0.0.0"),
  port = getOption("ambiorix.port", NULL),
  log = getOption("ambiorix.logger", FALSE)
)
```

Arguments:

`host` A string defining the host.

`port` Integer defining the port, defaults to `ambiorix.port` option: uses a random port if `NULL`.

`log` Whether to generate a log of events.

Details: Define the webserver.

Method `listen()`**:**

Usage:

```
Ambiorix$listen(port)
```

Arguments:

`port` Port number.

Details: Specifies the port to listen on.

Examples:

```
app <- Ambiorix$new()

app$listen(3000L)

app$get("/", function(req, res){
 res$send("Using {ambiorix}!")
})

if(interactive())
 app$start()
```

Method `get()`**:**

Usage:

```
Ambiorix$get(path, handler, error = NULL)
```

Arguments:

`path` Route to listen to, : defines a parameter.

`handler` Function that accepts the request and returns an object describing an httpuv response, e.g.: `response()`.

`error` Handler function to run on error.

Details: GET Method
Add routes to listen to.

Examples:

```
app <- Ambiorix$new()

app$get("/", function(req, res){
 res$send("Using {ambiorix}!")
})

if(interactive())
 app$start()
```

Method `put()`**:**

Usage:

```
Ambiorix$put(path, handler, error = NULL)
```

Arguments:

`path` Route to listen to, : defines a parameter.

`handler` Function that accepts the request and returns an object describing an httpuv response, e.g.: `response()`.

`error` Handler function to run on error.

Details: PUT Method
Add routes to listen to.

Method `patch()`**:**

Usage:

```
Ambiorix$patch(path, handler, error = NULL)
```

Arguments:

`path` Route to listen to, : defines a parameter.

`handler` Function that accepts the request and returns an object describing an httpuv response, e.g.: `response()`.

`error` Handler function to run on error.

Details: PATCH Method
Add routes to listen to.

Method `delete()`**:**

Usage:

```
Ambiorix$delete(path, handler, error = NULL)
```

Arguments:

`path` Route to listen to, `:` defines a parameter.

`handler` Function that accepts the request and returns an object describing an httpuv response, e.g.: `response()`.

`error` Handler function to run on error.

Details: DELETE Method
Add routes to listen to.

Method `post()`**:**

Usage:

```
Ambiorix$post(path, handler, error = NULL)
```

Arguments:

`path` Route to listen to.

`handler` Function that accepts the request and returns an object describing an httpuv response, e.g.: `response()`.

`error` Handler function to run on error.

Details: POST Method
Add routes to listen to.

Method `all()`**:**

Usage:

```
Ambiorix$all(path, handler, error = NULL)
```

Arguments:

`path` Route to listen to.

`handler` Function that accepts the request and returns an object describing an httpuv response, e.g.: `response()`.

`error` Handler function to run on error.

Details: All Methods
Add routes to listen to for all methods `GET`, `POST`, `PUT`, `DELETE`, and `PATCH`.

Method `set_404()`**:**

Usage:

```
Ambiorix$set_404(handler)
```

Arguments:

`handler` Function that accepts the request and returns an object describing an httpuv response, e.g.: `response()`.

Details: Sets the 404 page.

Examples:

```
app <- Ambiorix$new()

app$set_404(function(req, res){
 res$send("Nothing found here")
})
```

```
app$get("/", function(req, res){
 res$send("Using {ambiorix}!")
})

if(interactive())
 app$start()
```

Method `static()`**:**

Usage:

```
Ambiorix$static(path, uri = "www")
```

Arguments:

`path` Local path to directory of assets.

`uri` URL path where the directory will be available.

Details: Static directories

Method `start()`**:**

Usage:

```
Ambiorix$start(auto_stop = TRUE, open = interactive())
```

Arguments:

`auto_stop` Whether to automatically stop the server when the functon exits.

`open` Whether to open the app the browser.

Details: Start Start the webserver.

Examples:

```
app <- Ambiorix$new()

app$get("/", function(req, res){
 res$send("Using {ambiorix}!")
})

if(interactive())
 app$start()
```

Method `receive()`**:**

Usage:

```
Ambiorix$receive(name, handler)
```

Arguments:

`name` Name of message.

`handler` Function to run when message is received.

Details: Receive Websocket Message

Examples:

```
app <- Ambiorix$new()

app$get("/", function(req, res){
 res$send("Using {ambiorix}!")
})

app$receive("hello", function(msg, ws){
 print(msg) # print msg received

 # send a message back
 ws$send("hello", "Hello back! (sent from R)")
})

if(interactive())
 app$start()
```

Method `serialiser()`**:**

Usage:

```
Ambiorix$serialiser(handler)
```

Arguments:

`handler` Function to use to serialise. This function should accept two arguments: the object to serialise and `....`

Details: Define Serialiser

Examples:

```
app <- Ambiorix$new()

app$serialiser(function(data, ...){
 jsonlite::toJSON(x, ..., pretty = TRUE)
})

app$get("/", function(req, res){
 res$send("Using {ambiorix}!")
})

if(interactive())
 app$start()
```

Method `stop()`**:**

Usage:

```
Ambiorix$stop()
```

Details: Stop Stop the webserver.

Method `print()`**:**

Usage:

```
Ambiorix$print()
```

Details: Print

Method `use()`**:**

Usage:

```
Ambiorix$use(use)
```

Arguments:

`use` Either a router as returned by Router or a function to use as middleware. If a function is passed, it must accept two arguments (the request, and the response): this function will be executed every time the server receives a request. *Middleware may but does not have to return a response, unlike other methods such as* `get` Note that multiple routers and middlewares can be used.

Details: Use a router or middleware

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
app <- Ambiorix$new()

app$get("/", function(req, res){
 res$send("Using {ambiorix}!")
})

app$on_stop <- function(){
 cat("Bye!\n")
}

if(interactive())
 app$start()


## ------------------------------------------------
## Method `Ambiorix$listen`
## ------------------------------------------------

app <- Ambiorix$new()

app$listen(3000L)

app$get("/", function(req, res){
 res$send("Using {ambiorix}!")
})

if(interactive())
 app$start()

## ------------------------------------------------
```

```
## Method `Ambiorix$get`
## ------------------------------------------------

app <- Ambiorix$new()

app$get("/", function(req, res){
 res$send("Using {ambiorix}!")
})

if(interactive())
 app$start()

## ------------------------------------------------
## Method `Ambiorix$set_404`
## ------------------------------------------------

app <- Ambiorix$new()

app$set_404(function(req, res){
 res$send("Nothing found here")
})

app$get("/", function(req, res){
 res$send("Using {ambiorix}!")
})

if(interactive())
 app$start()

## ------------------------------------------------
## Method `Ambiorix$start`
## ------------------------------------------------

app <- Ambiorix$new()

app$get("/", function(req, res){
 res$send("Using {ambiorix}!")
})

if(interactive())
 app$start()

## ------------------------------------------------
## Method `Ambiorix$receive`
## ------------------------------------------------

app <- Ambiorix$new()

app$get("/", function(req, res){
 res$send("Using {ambiorix}!")
})

app$receive("hello", function(msg, ws){
```

```
 print(msg) # print msg received

 # send a message back
 ws$send("hello", "Hello back! (sent from R)")
})

if(interactive())
 app$start()

## ------------------------------------------------
## Method `Ambiorix$serialiser`
## ------------------------------------------------

app <- Ambiorix$new()

app$serialiser(function(data, ...){
 jsonlite::toJSON(x, ..., pretty = TRUE)
})

app$get("/", function(req, res){
 res$send("Using {ambiorix}!")
})

if(interactive())
 app$start()
```

`create_dockerfile` *Dockerfile*

Description

Create the dockerfile required to run the application. The dockerfile created will install packages from RStudio Public Package Manager which comes with pre-built binaries that much improve the speed of building of Dockerfiles.

Usage

```
create_dockerfile(port, host = "0.0.0.0")
```

Arguments

`port, host`	Port and host to serve the application.

Details

Reads the `DESCRIPTION` file of the project to produce the `Dockerfile`.

Examples

```
## Not run: create_dockerfile()
```

forward *Forward Method*

Description

Makes it such that the web server skips this method and uses the next one in line instead.

Usage

```
forward()
```

Value

An object of class `forward`.

Examples

```
app <- Ambiorix$new()

app$get("/next", function(req, res){
 forward()
})

app$get("/next", function(req, res){
 res$send("Hello")
})

if(interactive())
 app$start()
```

import *Import Files*

Description

Import all R-files in a directory.

Usage

```
import(...)
```

Arguments

`...`	Directory from which to import `.R` or `.r` files.

Value

Invisibly returns NULL.

Examples

```
## Not run: import("views")
```

Logger *Logger*

Description

Log events to ambiorix.log.

Details

The logger prepends every write with the current timestamp obtained with Sys.time(). Every write is a single line in the log.

Value

An object of class Logger that can be used to log events in an application.

Public fields

run Whether to actually log events to the file.

Methods

Public methods:

- Logger$new()
- Logger$write()
- Logger$log()
- Logger$print()
- Logger$clone()

Method new():

Usage:

```
Logger$new(log = TRUE)
```

Arguments:

log Whether to log events, if FALSE the write method does not have any effect.

Details: Initialise

Method write():

Usage:

```
Logger$write(label, ...)
```

Arguments:

`label` Label of event to log.

`...` Any other text to write alongside.

Details: Write events to log file - deprecated

Method `log()`**:**

Usage:

```
Logger$log(label, ...)
```

Arguments:

`label` Label of event to log.

`...` Any other text to write alongside.

Details: Write events to log file

Method `print()`**:**

Usage:

```
Logger$print()
```

Details: Print the logger state

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`new_log` *Logger*

Description

Returns a new logger using the `log` package.

Usage

```
new_log(prefix = ">", write = FALSE, file = "ambiorix.log", sep = "")
```

Arguments

`prefix`	String to prefix all log messages.
`write`	Whether to write the log to the `file`.
`file`	Name of the file to dump the logs to, only used if `write` is `TRUE`.
`sep`	Separator between `prefix` and other flags and messages.

Value

An R& of class `log::Logger`.

Examples

```
log <- new_log()
log$log("Hello world")
```

`parsers` *Parsers*

Description

Collection of parsers to translate request data.

Usage

```
parse_multipart(req, ...)

parse_json(req, ...)
```

Arguments

`req`	The request object.
`...`	Additional arguments passed to the internal parsers.

Value

Returns the parsed value as a `list`.

Functions

- `parse_multipart()`: Parse `multipart/form-data` using `mime::parse_multipart()`.
- `parse_json()`: Parse `multipart/form-data` using `jsonlite::fromJSON()`.

responses *Plain Responses*

Description

Plain HTTP Responses.

Usage

```
response(body, headers = list(`Content-Type` = "text/html"), status = 200L)

response_404(
  body = "404: Not found",
  headers = list(`Content-Type` = "text/html"),
  status = 404L
)

response_500(
  body = "500: Server Error",
  headers = list(`Content-Type` = "text/html"),
  status = 500L
)
```

Arguments

body	Body of response.
headers	HTTP headers.
status	Response status

Examples

```
app <- Ambiorix$new()

# html
app$get("/", function(req, res){
 res$send("hello!")
})

# text
app$get("/text", function(req, res){
 res$text("hello!")
})

if(interactive())
 app$start()
```

robj *Data Object*

Description

Treats a data element rendered in a response (`res$render`) as a data object and ultimately uses `dput()`.

Usage

```
robj(obj)
```

Arguments

`obj`	R object to treat.

Details

For instance in a template, x <- [% var %] will not work with `res$render(data=list(var = "hello"))` because this will be replace like `x <-hello` (missing quote): breaking the template. Using `robj` one would obtain `x <-"hello"`.

Router *Router*

Description

Web server.

Public fields

`error` 500 response when the route errors, must a handler function that accepts the request and the response, by default uses `response_500()`.

Methods

Public methods:

- `Router$new()`
- `Router$get()`
- `Router$put()`
- `Router$patch()`
- `Router$delete()`
- `Router$post()`
- `Router$receive()`
- `Router$print()`

- Router$routes()
- Router$receivers()
- Router$clone()

Method `new()`**:**

Usage:

```
Router$new(path)
```

Arguments:

`path` The base path of the router.

Details: Define the base route.

Method `get()`**:**

Usage:

```
Router$get(path, handler, error = NULL)
```

Arguments:

`path` Route to listen to, : defines a parameter.

`handler` Function that accepts the request and returns an object describing an httpuv response, e.g.: response().

`error` Handler function to run on error.

Details: GET Method
Add routes to listen to.

Method `put()`**:**

Usage:

```
Router$put(path, handler, error = NULL)
```

Arguments:

`path` Route to listen to, : defines a parameter.

`handler` Function that accepts the request and returns an object describing an httpuv response, e.g.: response().

`error` Handler function to run on error.

Details: PUT Method
Add routes to listen to.

Method `patch()`**:**

Usage:

```
Router$patch(path, handler, error = NULL)
```

Arguments:

`path` Route to listen to, : defines a parameter.

`handler` Function that accepts the request and returns an object describing an httpuv response, e.g.: response().

`error` Handler function to run on error.

Details: PATCH Method
Add routes to listen to.

Method `delete()`**:**

Usage:

```
Router$delete(path, handler, error = NULL)
```

Arguments:

`path` Route to listen to, : defines a parameter.

`handler` Function that accepts the request and returns an object describing an httpuv response, e.g.: `response()`.

`error` Handler function to run on error.

Details: DELETE Method
Add routes to listen to.

Method `post()`**:**

Usage:

```
Router$post(path, handler, error = NULL)
```

Arguments:

`path` Route to listen to.

`handler` Function that accepts the request and returns an object describing an httpuv response, e.g.: `response()`.

`error` Handler function to run on error.

Details: POST Method
Add routes to listen to.

Method `receive()`**:**

Usage:

```
Router$receive(name, handler)
```

Arguments:

`name` Name of message.

`handler` Function to run when message is received.

Details: Receive Websocket Message

Method `print()`**:**

Usage:

```
Router$print()
```

Details: Print

Method `routes()`**:**

Usage:

```
Router$routes()
```

Details: Get the routes

Method `receivers()`**:**

Usage:

```
Router$receivers()
```

Details: Get the receivers

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.

Examples

```
# log
logger <- new_log()
# router
# create router
router <- Router$new("/users")

router$get("/", function(req, res){
 res$send("List of users")
})

router$get("/:id", function(req, res){
 logger$log("Return user id:", req$params$id)
 res$send(req$params$id)
})

router$get("/:id/profile", function(req, res){
 msg <- sprintf("This is the profile of user #%s", req$params$id)
 res$send(msg)
})

# core app
app <- Ambiorix$new()

app$get("/", function(req, res){
 res$send("Home!")
})

# mount the router
app$use(router)

if(interactive())
 app$start()
```

set_params *Set Parameters*

Description

Set the query's parameters.

Usage

```
set_params(path, route = NULL)
```

Arguments

path	Correspond's the the requests' PATH_INFO
route	See Route

Value

Parameter list

stop_all *Stop*

Description

Stop all servers.

Usage

```
stop_all()
```

websocket_client *Websocket Client*

Description

Handle ambiorix websocket client.

Usage

```
copy_websocket_client(path)

get_websocket_client()
```

Arguments

`path`	Path to copy the file to.

Functions

- `copy_websocket_client` Copies the websocket client file, useful when ambiorix was not setup with the ambiorix generator.
- `get_websocket_client` Retrieves the full path to the local websocket client.

Index