Package ‘usethat’

September 20, 2021

Type Package

Title Automate Analytic Project Setup and Development

URL https://tidylab.github.io/usethat/,
https://github.com/tidylab/usethat

BugReports https://github.com/tidylab/usethat/issues

Version 0.3.0

Date 2021-09-01

Maintainer Harel Lustiger <tidylab@gmail.com>

Description Automate analytic project setup tasks that are otherwise performed manually. This includes setting up docker, spinning up a microservice, and more.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.1.1

Language en-GB

Depends R (>= 3.5)

Suggests testthat

Imports microservices (>= 0.1.2), purrr, usethis, withr

NeedsCompilation no

Author Harel Lustiger [aut, cre] (<https://orcid.org/0000-0003-2953-9598>)

Repository CRAN

Date/Publication 2021-09-20 09:20:02 UTC

R topics documented:

add_service *Add a Service Route to the Microservice*

Description

Add a Service Route to the Microservice

Usage

```
add_service(path = ".", name, overwrite = FALSE)
```

Arguments

`path`	(`character`) Where is the project root folder?
`name`	(`character`) what is the service route name? For example, if `name` = "repository" then the set of services would become available at http://127.0.0.1:8080/repository/.
`overwrite`	(`logical`) Should existing destination files be overwritten?

Details

Lay the infrastructure for an additional set of services. That includes adding a unit test, adding an endpoint, and extending the entrypointy.

Note: `add_service` adds a service to pre-existing plumber microservice which you could deploy by calling `use_microservice`.

How It Works:

Given a `path` (.) to a folder and a `name` (repository)

When `add_service` is called

Then the function creates the following files:

```
tests/testthat/test-endpoint-plumber-repository.R
inst/endpoints/plumber-repository.R
```

And updates the following files:

```
inst/entrypoints/plumber-foreground.R
```

When to Use:

In scenarios where services are thematically linked to each other. Examples for themes that should be mounted separately:

- â€˜forecastingâ€™ and â€˜anomaly detectionâ€™
- â€˜userâ€™ and â€˜businessâ€™

Value

No return value, called for side effects.

See Also

Other microservice utilities: `use_microservice()`

Examples

```
path <- tempfile()
dir.create(path, showWarnings = FALSE, recursive = TRUE)
use_microservice(path)

add_service(path, name = "repository")

list.files(path, recursive = TRUE)
```

`use_microservice` *Use a plumber Microservice in an R Project*

Description

Use a plumber Microservice in an R Project

Usage

```
use_microservice(path = ".", overwrite = FALSE)
```

Arguments

`path`	(`character`) Where is the project root folder?
`overwrite`	(`logical`) Should existing destination files be overwritten?

Details

How It Works:

Given a `path` to a folder
When `use_microservice(path = ".")` is called
Then the function creates the following files:

```
tests/testthat/test-endpoint-plumber-utility.R
inst/configurations/plumber.yml
inst/endpoints/plumber-utility.R
inst/entrypoints/plumber-background.R
inst/entrypoints/plumber-foreground.R
```

And updates the following files:

```
tests/testthat/helpers-xyz.R
```

And adds the following packages to the DESCRIPTION file:

type	package	version
Suggests	config	*
Suggests	httptest	*
Suggests	httr	*
Imports	jsonlite	*
Suggests	pkgload	*
Suggests	plumber	>= 1.0.0
Imports	purrr	*
Suggests	testthat	*
Suggests	usethis	*
Suggests	promises	*
Suggests	future	*

When to Use `plumber`:

- A Single user/machine applications.
- Scheduled tasks. For example, you could use AirFlow with HTTP Operators to automate processes.

`plumber` *Advantages:*

- Comes with familiar way to document the microservice endpoint.
- Maturing package that comes with documentation, examples and support.

`plumber` *Disadvantages:*

- Runs on a single thread. That means that parallel algorithms such as random forest, can only be run on one core.
- Serves only one caller at a time.
- Canâ€™t make inward calls for other services, That means plumber canâ€™t be re-entrant. For example, if a microservice has three endpoints,`read_table`, `write_table`, and `orchestrator`, where the `orchestrator` reads a data table, transforms it, and writes it back, then the `orchestrator` canâ€™t make inwards calls via HTTP to `read_table` and `write_table`.

Note: While `plumber` is single-threaded by nature, it is possible to perform parallel execution using the `promises` package. See links under References.

Workflow:

1. Deploy the Microservice infrastructure

```
microservices::use_microservice(path = ".")
remotes::install_deps()
devtools::document()
```

1. Spin-up the microservice by running `source("./inst/entrypoints/plumber-background.R")`
2. Run the microservice unit-test by pressing Ctrl+Shift+T on Windows

Congratulations! You have added a microservice to your application and tested that it works.

References:

- Parallel execution in plumber
- `promises` package

Value

No return value, called for side effects.

See Also

Other microservice utilities: `add_service()`

Examples

```
path <- tempfile()
use_microservice(path)

list.files(path, recursive = TRUE)

cat(read.dcf(file.path(path, "DESCRIPTION"), "Imports"))
cat(read.dcf(file.path(path, "DESCRIPTION"), "Suggests"))
```

`use_na` *Use NA of different classes in your project*

Description

R has several built-in NA values that correspond to the atomic data types, such as `NA` (logical), `NA_integer_` and `NA_character_`. Calling `use_na()` allows the programmer to have NA values of any class. In addition, `use_na()` provides several useful NA values such as `NA_list_`, `NA_Date_` and `NA_POSIXct_`.

Usage

```
use_na(path = "R", export = TRUE)
```

Arguments

`path`	(`character`) A path pointing at where to copy the file.
`export`	If `TRUE`, the file content is exported to `NAMESPACE`.

Details

The function copies a file with several NA values to '`path`/utils-na.R'.

Value

No return value, called for side effects.

Examples

```
path <- tempfile()
use_na(path)
print(readLines(file.path(path, "utils-na.R")))
```

`use_pipes` *Use different pipes in your package*

Description

The function adds the useful operators to use in your project. These operators include:

- %>% Forward Pipe operator
- %||% NULL operator

Usage

```
use_pipes(path = "R", export = TRUE)
```

Arguments

`path`	(`character`) A path pointing at where to copy the file.
`export`	If `TRUE`, the file content is exported to `NAMESPACE`.

Details

The function:

1. Copies a file with several pipes '`path`/utils-pipes.R' and
2. Imports the `purrr`package in the project DESCRIPTION file

Value

No return value, called for side effects.

Examples

```
path <- tempfile()
use_pipes(path)
print(readLines(file.path(path, "utils-pipes.R")))
```

Index