Package: teal (via r-universe)

September 4, 2024

Type Package

Title Exploratory Web Apps for Analyzing Clinical Trials Data

Version 0.15.2

Date 2024-03-07

Description A 'shiny' based interactive exploration framework for analyzing clinical trials data. 'teal' currently provides a dynamic filtering facility and different data viewers. 'teal' 'shiny' applications are built using standard 'shiny' modules.

License Apache License 2.0

URL https://insightsengineering.github.io/teal/, https://github.com/insightsengineering/teal/

BugReports https://github.com/insightsengineering/teal/issues

Depends R (>= 4.0), shiny (>= 1.7.0), teal.data (>= 0.4.0), teal.slice (>= 0.5.0)

Imports checkmate (>= 2.1.0), jsonlite, lifecycle (>= 0.2.0), logger (>= 0.2.0), magrittr (>= 1.5), methods, rlang (>= 1.0.0), shinyjs, stats, teal.code (>= 0.5.0), teal.logger (>= 0.1.1), teal.reporter (>= 0.2.0), teal.widgets (>= 0.4.0), utils

Suggests bslib, knitr (>= 1.42), MultiAssayExperiment, R6, rmarkdown (>= 2.19), shinyvalidate, testthat (>= 3.1.5), withr (>= 2.1.0), yaml (>= 1.1.0)

VignetteBuilder knitr

RdMacros lifecycle

Config/Needs/verdepcheck rstudio/shiny, insightsengineering/teal.data, insightsengineering/teal.slice, mllg/checkmate, jeroen/jsonlite, r-lib/lifecycle, daroczig/logger, tidyverse/magrittr, r-lib/rlang, daattali/shinyjs, insightsengineering/teal.logger, insightsengineering/teal.reporter, insightsengineering/teal.widgets, rstudio/bslib, yihui/knitr, bioc::MultiAssayExperiment, r-lib/R6, rstudio/rmarkdown,

rstudio/shinyvalidate, insightsengineering/teal.code, r-lib/testthat, r-lib/withr, yaml=vubiostat/r-yaml

Config/Needs/website insightsengineering/nesttemplate

Encoding UTF-8

Language en-US

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Collate 'dummy_functions.R' 'get_rcode_utils.R' 'include_css_js.R' 'modules.R' 'init.R' 'landing_popup_module.R' 'module_filter_manager.R' 'module_nested_tabs.R' 'module_snapshot_manager.R' 'module_tabs_with_filters.R' 'module_teal.R' 'module_teal_with_splash.R' 'reporter_previewer_module.R' 'show_rcode_modal.R' 'tdata.R' 'teal.R' 'teal_data_module.R' 'teal_data_module-eval_code.R' 'teal_data_module-within.R' 'teal_reporter.R' 'teal_slices-store.R' 'teal_slices.R' 'utils.R' 'validate_inputs.R' 'validations.R' 'zzz.R'

Repository https://insightsengineering.r-universe.dev

RemoteUrl https://github.com/insightsengineering/teal

RemoteRef v0.15.2

RemoteSha 06a0a581a132590e85cb147d9ab1853558ab150e

Contents

`as_tdata` *Downgrade* `teal_data` *objects in modules for compatibility*

Description

Convert `teal_data` to `tdata` in `teal` modules.

Usage

```
as_tdata(x)
```

Arguments

`x`	data object, either `tdata` or `teal_data`, the latter possibly in a reactive expression

Details

Recent changes in `teal` cause modules to fail because modules expect a `tdata` object to be passed to the `data` argument but instead they receive a `teal_data` object, which is additionally wrapped in a reactive expression in the server functions. In order to easily adapt such modules without a proper refactor, use this function to downgrade the `data` argument.

Value

Object of class `tdata`.

Examples

```
td <- teal_data()
td <- within(td, iris <- iris) %>% within(mtcars <- mtcars)
td
as_tdata(td)
as_tdata(reactive(td))
```

build_app_title *Build app title with favicon*

Description

A helper function to create the browser title along with a logo.

Usage

```
build_app_title(
  title = "teal app",
  favicon =
   "https://raw.githubusercontent.com/insightsengineering/hex-stickers/main/PNG/nest.png"
)
```

Arguments

title	(character) The browser title for the teal app.
favicon	(character) The path for the icon for the title. The image/icon path can be remote or the static path accessible by shiny, like the www/

Value

A shiny.tag containing the element that adds the title and logo to the shiny app.

example_module *An example* teal *module*

Description

[Experimental]

Usage

```
example_module(label = "example teal module", datanames = "all")
```

Arguments

label	(character(1)) Label shown in the navigation item for the module or module group. For modules() defaults to "root". See Details.
datanames	(character) A vector with datanames that are relevant for the item. The filter panel will automatically update the shown filters to include only filters in the listed datasets. NULL will hide the filter panel, and the keyword "all" will show filters of all datasets. datanames also determines a subset of datasets which are appended to the data argument in server function.

Value

A `teal` module which can be included in the `modules` argument to `init()`.

Examples

```
app <- init(
  data = teal_data(IRIS = iris, MTCARS = mtcars),
  modules = example_module()
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

---

get_code_tdata *Wrapper for* `get_code.tdata`

Description

This wrapper is to be used by downstream packages to extract the code of a `tdata` object.

Usage

```
get_code_tdata(data)
```

Arguments

`data`	(`tdata`) object

Value

(`character`) code used in the `tdata` object.

---

get_metadata *Function to get metadata from a* `tdata` *object*

Description

Function to get metadata from a `tdata` object

Usage

```
get_metadata(data, dataname)

## S3 method for class 'tdata'
get_metadata(data, dataname)

## Default S3 method:
get_metadata(data, dataname)
```

Arguments

data	(tdata - object) to extract the data from
dataname	(character(1)) the dataset name whose metadata is requested

Value

Either list of metadata or NULL if no metadata.

---

init *Create the server and UI function for the* shiny *app*

Description

[Stable]

End-users: This is the most important function for you to start a teal app that is composed of teal modules.

Usage

```
init(
  data,
  modules,
  filter = teal_slices(),
  title = build_app_title(),
  header = tags$p(),
  footer = tags$p(),
  id = character(0)
)
```

Arguments

data	(teal_data or teal_data_module) For constructing the data object, refer to teal_data() and teal_data_module().
modules	(list or teal_modules or teal_module) nested list of teal_modules or teal_module objects or a single teal_modules or teal_module object. These are the specific output modules which will be displayed in the teal application. See modules() and module() for more details.
filter	(teal_slices) Specifies the initial filter using teal_slices().
title	(shiny.tag or character(1)) The browser window title. Defaults to a title "teal app" with the icon of NEST. Can be created using the build_app_title() or by passing a valid shiny.tag which is a head tag with title and link tag.
header	(shiny.tag or character(1)) The header of the app.
footer	(shiny.tag or character(1)) The footer of the app.
id	(character) Optional string specifying the shiny module id in cases it is used as a shiny module rather than a standalone shiny app. This is a legacy feature.

Details

When initializing the `teal` app, if `datanames` are not set for the `teal_data` object, defaults from the `teal_data` environment will be used.

Value

Named list with server and UI functions.

Examples

```
app <- init(
  data = teal_data(
    new_iris = transform(iris, id = seq_len(nrow(iris))),
    new_mtcars = transform(mtcars, id = seq_len(nrow(mtcars))),
    code = "
      new_iris <- transform(iris, id = seq_len(nrow(iris)))
      new_mtcars <- transform(mtcars, id = seq_len(nrow(mtcars)))
    "
  ),
  modules = modules(
    module(
      label = "data source",
      server = function(input, output, session, data) {},
      ui = function(id, ...) div(p("information about data source")),
      datanames = "all"
    ),
    example_module(label = "example teal module"),
    module(
      "Iris Sepal.Length histogram",
      server = function(input, output, session, data) {
        output$hist <- renderPlot(
          hist(data()[["new_iris"]]$Sepal.Length)
        )
      },
      ui = function(id, ...) {
        ns <- NS(id)
        plotOutput(ns("hist"))
      },
      datanames = "new_iris"
    )
  ),
  filter = teal_slices(
    teal_slice(dataname = "new_iris", varname = "Species"),
    teal_slice(dataname = "new_iris", varname = "Sepal.Length"),
    teal_slice(dataname = "new_mtcars", varname = "cyl"),
    exclude_varnames = list(new_iris = c("Sepal.Width", "Petal.Width")),
    module_specific = TRUE,
    mapping = list(
      `example teal module` = "new_iris Species",
      `Iris Sepal.Length histogram` = "new_iris Species",
      global_filters = "new_mtcars cyl"
    )
```

```
  ),
  title = "App title",
  header = tags$h1("Sample App"),
  footer = tags$p("Copyright 2017 - 2023")
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

`join_keys.tdata` *Extract* `join_keys` *from* `tdata`

Description

Extract `join_keys` from `tdata`

Usage

```
## S3 method for class 'tdata'
join_keys(data, ...)
```

Arguments

`data`	(`tdata`) object
`...`	Additional arguments (not used)

`landing_popup_module` *Landing popup module*

Description

Creates a landing welcome popup for `teal` applications.

This module is used to display a popup dialog when the application starts. The dialog blocks access to the application and must be closed with a button before the application can be viewed.

Usage

```
landing_popup_module(
  label = "Landing Popup",
  title = NULL,
  content = NULL,
  buttons = modalButton("Accept")
)
```

Arguments

label	(character(1)) Label of the module.
title	(character(1)) Text to be displayed as popup title.
content	(character(1), shiny.tag or shiny.tag.list) with the content of the popup. Passed to ... of shiny::modalDialog. See examples.
buttons	(shiny.tag or shiny.tag.list) Typically a modalButton or actionButton. See examples.

Value

A teal_module (extended with teal_landing_module class) to be used in teal applications.

Examples

```
app1 <- init(
  data = teal_data(iris = iris),
  modules = modules(
    landing_popup_module(
      content = "A place for the welcome message or a disclaimer statement.",
      buttons = modalButton("Proceed")
    ),
    example_module()
  )
)
if (interactive()) {
  shinyApp(app1$ui, app1$server)
}

app2 <- init(
  data = teal_data(iris = iris),
  modules = modules(
    landing_popup_module(
      title = "Welcome",
      content = tags$b(
        "A place for the welcome message or a disclaimer statement.",
        style = "color: red;"
      ),
      buttons = tagList(
        modalButton("Proceed"),
        actionButton("read", "Read more",
          onclick = "window.open('http://google.com', '_blank')"
        ),
        actionButton("close", "Reject", onclick = "window.close()")
      )
    ),
    example_module()
  )
)

if (interactive()) {
  shinyApp(app2$ui, app2$server)
```

```
}
```

`module_teal_with_splash` *Add splash screen to* `teal` *application*

Description

[Stable]

Displays custom splash screen during initial delayed data loading.

Usage

```
ui_teal_with_splash(
  id,
  data,
  title = build_app_title(),
  header = tags$p(),
  footer = tags$p()
)

srv_teal_with_splash(id, data, modules, filter = teal_slices())
```

Arguments

`id`	(`character(1)`) module id
`data`	(`teal_data` or `teal_data_module`) For constructing the data object, refer to `teal_data()` and `teal_data_module()`.
`title`	(`shiny.tag` or `character(1)`) The browser window title. Defaults to a title "teal app" with the icon of NEST. Can be created using the `build_app_title()` or by passing a valid `shiny.tag` which is a head tag with title and link tag.
`header`	(`shiny.tag` or `character(1)`) The header of the app.
`footer`	(`shiny.tag` or `character(1)`) The footer of the app.
`modules`	(`teal_modules`) object containing the output modules which will be displayed in the `teal` application. See `modules()` and `module()` for more details.
`filter`	(`teal_slices`) Specifies the initial filter using `teal_slices()`.

Details

This module pauses app initialization pending delayed data loading. This is necessary because the filter panel and modules depend on the data to initialize.

`teal_with_splash` follows the `shiny` module convention. `init()` is a wrapper around this that assumes that `teal` it is the top-level module and cannot be embedded.

Note: It is no longer recommended to embed `teal` in `shiny` apps as a module. but rather use `init` to create a standalone application.

Value

Returns a `reactive` expression containing a `teal_data` object when data is loaded or `NULL` when it is not.

See Also

`init()`

Examples

```
teal_modules <- modules(example_module())
# Shiny app with modular integration of teal
ui <- fluidPage(
  ui_teal_with_splash(id = "app1", data = teal_data())
)

server <- function(input, output, session) {
  srv_teal_with_splash(
    id = "app1",
    data = teal_data(iris = iris),
    modules = teal_modules
  )
}

if (interactive()) {
  shinyApp(ui, server)
}
```

`reporter_previewer_module`

Create a `teal` *module for previewing a report*

Description

[Experimental]

This function wraps `teal.reporter::reporter_previewer_ui()` and `teal.reporter::reporter_previewer_srv()` into a `teal_module` to be used in `teal` applications.

If you are creating a `teal` application using `init()` then this module will be added to your application automatically if any of your `teal_modules` support report generation.

Usage

```
reporter_previewer_module(label = "Report previewer", server_args = list())
```

Arguments

label	(character(1)) Label shown in the navigation item for the module or module group. For modules() defaults to "root". See Details.
server_args	(named list) Arguments passed to teal.reporter::reporter_previewer_srv().

Value

teal_module (extended with teal_module_previewer class) containing the teal.reporter previewer functionality.

---

report_card_template *Template function for* TealReportCard *creation and customization*

Description

This function generates a report card with a title, an optional description, and the option to append the filter state list.

Usage

```
report_card_template(
  title,
  label,
  description = NULL,
  with_filter,
  filter_panel_api
)
```

Arguments

title	(character(1)) title of the card (unless overwritten by label)
label	(character(1)) label provided by the user when adding the card
description	(character(1)) optional additional description
with_filter	(logical(1)) flag indicating to add filter state
filter_panel_api	(FilterPanelAPI) object with API that allows the generation of the filter state in the report

Value

(TealReportCard) populated with a title, description and filter state.

show_rcode_modal *Show* R *code modal*

Description

[Stable]

Use the `shiny::showModal()` function to show the R code inside.

Usage

```
show_rcode_modal(title = NULL, rcode, session = getDefaultReactiveDomain())
```

Arguments

title	(character(1)) Title of the modal, displayed in the first comment of the R code.
rcode	(character) vector with R code to show inside the modal.
session	(ShinySession optional) shiny session object, if missing then `shiny::getDefaultReactiveDomain()` is used.

References

`shiny::showModal()`

tdata *Create a* tdata *object*

Description

[Deprecated]

Create a new object called tdata which contains data, a reactive list of data.frames (or MultiAssayExperiment), with attributes:

- code (reactive) containing code used to generate the data
- join_keys (join_keys) containing the relationships between the data
- metadata (named list) containing any metadata associated with the data frames

Usage

```
new_tdata(data, code = "", join_keys = NULL, metadata = NULL)
```

Arguments

data	(named list) A list of data.frame or MultiAssayExperiment objects, which optionally can be reactive. Inside this object all of these items will be made reactive.
code	(character or reactive which evaluates to a character) containing the code used to generate the data. This should be reactive if the code is changing during a reactive context (e.g. if filtering changes the code). Inside this object code will be made reactive
join_keys	(teal.data::join_keys) object containing relationships between the datasets.
metadata	(named list) each element contains a list of metadata about the named data.frame Each element of these list should be atomic and length one.

Value

A tdata object.

See Also

as_tdata

Examples

```
data <- new_tdata(
  data = list(iris = iris, mtcars = reactive(mtcars), dd = data.frame(x = 1:10)),
  code = "iris <- iris
    mtcars <- mtcars
    dd <- data.frame(x = 1:10)",
  metadata = list(dd = list(author = "NEST"), iris = list(version = 1))
)

# Extract a data.frame
isolate(data[["iris"]]())

# Get code
isolate(get_code_tdata(data))

# Get metadata
get_metadata(data, "iris")
```

---

tdata2env *Function to convert a* tdata *object to an* environment

---

Description

Any reactive expressions inside tdata are evaluated first.

Usage

```
tdata2env(data)
```

Arguments

data	(tdata) object

Value

An environment.

Examples

```
data <- new_tdata(
  data = list(iris = iris, mtcars = reactive(mtcars)),
  code = "iris <- iris
    mtcars = mtcars"
)

my_env <- isolate(tdata2env(data))
```

TealReportCard TealReportCard

Description

[Experimental] Child class of ReportCard that is used for teal specific applications. In addition to the parent methods, it supports rendering teal specific elements such as the source code, the encodings panel content and the filter panel content as part of the meta data.

Super class

teal.reporter::ReportCard -> TealReportCard

Methods

Public methods:

- TealReportCard$append_src()
- TealReportCard$append_fs()
- TealReportCard$append_encodings()
- TealReportCard$clone()

Method append_src(): Appends the source code to the content meta data of this TealReportCard.

Usage:

```
TealReportCard$append_src(src, ...)
```

Arguments:

`src` (`character(1)`) code as text.

`...` any `rmarkdown` `R` chunk parameter and its value. But `eval` parameter is always set to `FALSE`.

Returns: Object of class `TealReportCard`, invisibly.

Examples:

```
card <- TealReportCard$new()$append_src(
  "plot(iris)"
)
card$get_content()[[1]]$get_content()
```

Method `append_fs()`**:** Appends the filter state list to the `content` and `metadata` of this `TealReportCard`. If the filter state list has an attribute named `formatted`, it appends it to the card otherwise it uses the default `yaml::as.yaml` to format the list. If the filter state list is empty, nothing is appended to the `content`.

Usage:

```
TealReportCard$append_fs(fs)
```

Arguments:

`fs` (`teal_slices`) object returned from `teal_slices()` function.

Returns: `self`, invisibly.

Method `append_encodings()`**:** Appends the encodings list to the `content` and `metadata` of this `TealReportCard`.

Usage:

```
TealReportCard$append_encodings(encodings)
```

Arguments:

`encodings` (`list`) list of encodings selections of the `teal` app.

Returns: `self`, invisibly.

Examples:

```
card <- TealReportCard$new()$append_encodings(list(variable1 = "X"))
card$get_content()[[1]]$get_content()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `TealReportCard$append_src`
## ------------------------------------------------
```

```
card <- TealReportCard$new()$append_src(
  "plot(iris)"
)
card$get_content()[[1]]$get_content()

## ------------------------------------------------
## Method `TealReportCard$append_encodings`
## ------------------------------------------------

card <- TealReportCard$new()$append_encodings(list(variable1 = "X"))
card$get_content()[[1]]$get_content()
```

`teal_data_module` *Data module for* `teal` *applications*

Description

[Experimental]

Create a `teal_data_module` object and evaluate code on it with history tracking.

Usage

```
teal_data_module(ui, server)

## S4 method for signature 'teal_data_module,character'
eval_code(object, code)

## S3 method for class 'teal_data_module'
within(data, expr, ...)
```

Arguments

`ui`	(`function(id)`) `shiny` module UI function; must only take `id` argument
`server`	(`function(id)`) `shiny` module server function; must only take `id` argument; must return reactive expression containing `teal_data` object
`object`	(`teal_data_module`)
`code`	(`character` or `language`) code to evaluate. If `character`, comments are retained.
`data`	(`teal_data_module`) object
`expr`	(`expression`) to evaluate. Must be inline code. See
`...`	See `Details`.

Details

teal_data_module creates a shiny module to supply or modify data in a teal application. The module allows for running data pre-processing code (creation *and* some modification) after the app starts. The body of the server function will be run in the app rather than in the global environment. This means it will be run every time the app starts, so use sparingly.

Pass this module instead of a teal_data object in a call to init(). Note that the server function must always return a teal_data object wrapped in a reactive expression.

See vignette vignette("data-as-shiny-module", package = "teal") for more details.

eval_code evaluates given code in the environment of the teal_data object created by the teal_data_module. The code is added to the @code slot of the teal_data.

within is a convenience function for evaluating inline code inside the environment of a teal_data_module. It accepts only inline expressions (both simple and compound) and allows for injecting values into expr through the ... argument: as name:value pairs are passed to ..., name in expr will be replaced with value.

Value

teal_data_module returns an object of class teal_data_module.

eval_code returns a teal_data_module object with a delayed evaluation of code when the module is run.

within returns a teal_data_module object with a delayed evaluation of expr when the module is run.

See Also

teal.data::teal_data, teal.code::qenv()

Examples

```
tdm <- teal_data_module(
  ui = function(id) {
    ns <- NS(id)
    actionButton(ns("submit"), label = "Load data")
  },
  server = function(id) {
    moduleServer(id, function(input, output, session) {
      eventReactive(input$submit, {
        data <- within(
          teal_data(),
          {
            dataset1 <- iris
            dataset2 <- mtcars
          }
        )
        datanames(data) <- c("dataset1", "dataset2")

        data
      })
```

```
    })
  }
)

eval_code(tdm, "dataset1 <- subset(dataset1, Species == 'virginica')")

within(tdm, dataset1 <- subset(dataset1, Species == "virginica"))

# use additional parameter for expression value substitution.
valid_species <- "versicolor"
within(tdm, dataset1 <- subset(dataset1, Species %in% species), species = valid_species)
```

`teal_modules` *Create* `teal_module` *and* `teal_modules` *objects*

Description

[Stable]

Create a nested tab structure to embed modules in a `teal` application.

Usage

```
module(
  label = "module",
  server = function(id, ...) {
     moduleServer(id, function(input, output, session) {

       })
 },
  ui = function(id, ...) {
     tags$p(paste0("This module has no UI (id: ", id, " )"))

    },
  filters,
  datanames = "all",
  server_args = NULL,
  ui_args = NULL
)

modules(..., label = "root")

## S3 method for class 'teal_module'
format(x, indent = 0, ...)

## S3 method for class 'teal_module'
print(x, ...)
```

```
## S3 method for class 'teal_modules'
format(x, indent = 0, ...)

## S3 method for class 'teal_modules'
print(x, ...)
```

Arguments

label	(character(1)) Label shown in the navigation item for the module or module group. For modules() defaults to "root". See Details.
server	(function) shiny module with following arguments: • id - teal will set proper shiny namespace for this module (see shiny::moduleServer()). • input, output, session - (optional; not recommended) When provided, then shiny::callModule() will be used to call a module. From shiny 1.5.0, the recommended way is to use shiny::moduleServer() instead which doesn't require these arguments. • data (optional) When provided, the module will be called with teal_data object (i.e. a list of reactive (filtered) data specified in the filters argument) as the value of this argument. • datasets (optional) When provided, the module will be called with FilteredData object as the value of this argument. (See teal.slice::FilteredData). • reporter (optional) When provided, the module will be called with Reporter object as the value of this argument. (See teal.reporter::Reporter). • filter_panel_api (optional) When provided, the module will be called with FilterPanelAPI object as the value of this argument. (See teal.slice::FilterPanelAPI). • ... (optional) When provided, server_args elements will be passed to the module named argument or to the ....
ui	(function) shiny UI module function with following arguments: • id - teal will set proper shiny namespace for this module. • ... (optional) When provided, ui_args elements will be passed to the module named argument or to the ....
filters	(character) Deprecated. Use datanames instead.
datanames	(character) A vector with datanames that are relevant for the item. The filter panel will automatically update the shown filters to include only filters in the listed datasets. NULL will hide the filter panel, and the keyword "all" will show filters of all datasets. datanames also determines a subset of datasets which are appended to the data argument in server function.
server_args	(named list) with additional arguments passed on to the server function.
ui_args	(named list) with additional arguments passed on to the UI function.
...	• For modules(): (teal_module or teal_modules) Objects to wrap into a tab. • For format() and print(): Arguments passed to other methods.
x	(teal_module or teal_modules) Object to format/print.
indent	(integer(1)) Indention level; each nested element is indented one level more.

Details

module() creates an instance of a teal_module that can be placed in a teal application. modules() shapes the structure of a the application by organizing teal_module within the navigation panel. It wraps teal_module and teal_modules objects in a teal_modules object, which results in a nested structure corresponding to the nested tabs in the final application.

Note that for modules() label comes after ..., so it must be passed as a named argument, otherwise it will be captured by ....

The labels "global_filters" and "Report previewer" are reserved because they are used by the mapping argument of teal_slices() and the report previewer module reporter_previewer_module(), respectively.

Value

module() returns an object of class teal_module.

modules() returns a teal_modules object which contains following fields:

- label: taken from the label argument.
- children: a list containing objects passed in .... List elements are named after their label attribute converted to a valid shiny id.

Examples

```
library(shiny)

module_1 <- module(
  label = "a module",
  server = function(id, data) {
    moduleServer(
      id,
      module = function(input, output, session) {
        output$data <- renderDataTable(data()[["iris"]])
      }
    )
  },
  ui = function(id) {
    ns <- NS(id)
    tagList(dataTableOutput(ns("data")))
  },
  datanames = "all"
)

module_2 <- module(
  label = "another module",
  server = function(id) {
    moduleServer(
      id,
      module = function(input, output, session) {
        output$text <- renderText("Another Module")
      }
    )
```

```
  },
  ui = function(id) {
    ns <- NS(id)
    tagList(textOutput(ns("text")))
  },
  datanames = NULL
)

modules <- modules(
  label = "modules",
  modules(
    label = "nested modules",
    module_1
  ),
  module_2
)

app <- init(
  data = teal_data(iris = iris),
  modules = modules
)

if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

`validate_has_data` *Validate that dataset has a minimum number of observations*

Description

[Stable]

Usage

```
validate_has_data(
  x,
  min_nrow = NULL,
  complete = FALSE,
  allow_inf = TRUE,
  msg = NULL
)
```

Arguments

`x`	(`data.frame`)
`min_nrow`	(`numeric(1)`) Minimum allowed number of rows in `x`.
`complete`	(`logical(1)`) Flag specifying whether to check only complete cases. Defaults to `FALSE`.

allow_inf	(logical(1)) Flag specifying whether to allow infinite values. Defaults to TRUE.
msg	(character(1)) Additional message to display alongside the default message.

Details

This function is a wrapper for shiny::validate.

Examples

```
library(teal)
ui <- fluidPage(
  sliderInput("len", "Max Length of Sepal",
    min = 4.3, max = 7.9, value = 5
  ),
  plotOutput("plot")
)

server <- function(input, output) {
  output$plot <- renderPlot({
    iris_df <- iris[iris$Sepal.Length <= input$len, ]
    validate_has_data(
      iris_df,
      min_nrow = 10,
      complete = FALSE,
      msg = "Please adjust Max Length of Sepal"
    )

    hist(iris_df$Sepal.Length, breaks = 5)
  })
}
if (interactive()) {
  shinyApp(ui, server)
}
```

---

validate_has_elements *Validates that vector has length greater than 0*

Description

[Stable]

Usage

```
validate_has_elements(x, msg)
```

Arguments

x	vector
msg	message to display

Details

This function is a wrapper for `shiny::validate`.

Examples

```
data <- data.frame(
  id = c(1:10, 11:20, 1:10),
  strata = rep(c("A", "B"), each = 15)
)
ui <- fluidPage(
  selectInput("ref1", "Select strata1 to compare",
    choices = c("A", "B", "C"), selected = "A"
  ),
  selectInput("ref2", "Select strata2 to compare",
    choices = c("A", "B", "C"), selected = "B"
  ),
  verbatimTextOutput("arm_summary")
)

server <- function(input, output) {
  output$arm_summary <- renderText({
    sample_1 <- data$id[data$strata == input$ref1]
    sample_2 <- data$id[data$strata == input$ref2]

    validate_has_elements(sample_1, "No subjects in strata1.")
    validate_has_elements(sample_2, "No subjects in strata2.")

    paste0(
      "Number of samples in: strata1=", length(sample_1),
      " comparions strata2=", length(sample_2)
    )
  })
}
if (interactive()) {
  shinyApp(ui, server)
}
```

`validate_has_variable` *Validates that dataset contains specific variable*

Description

[Stable]

Usage

```
validate_has_variable(data, varname, msg)
```

Arguments

data	(data.frame)
varname	(character(1)) name of variable to check for in data
msg	(character(1)) message to display if data does not include varname

Details

This function is a wrapper for shiny::validate.

Examples

```
data <- data.frame(
  one = rep("a", length.out = 20),
  two = rep(c("a", "b"), length.out = 20)
)
ui <- fluidPage(
  selectInput(
    "var",
    "Select variable",
    choices = c("one", "two", "three", "four"),
    selected = "one"
  ),
  verbatimTextOutput("summary")
)

server <- function(input, output) {
  output$summary <- renderText({
    validate_has_variable(data, input$var)
    paste0("Selected treatment variables: ", paste(input$var, collapse = ", "))
  })
}
if (interactive()) {
  shinyApp(ui, server)
}
```

---

validate_in *Validates that vector includes all expected values*

Description

[Stable]

Usage

```
validate_in(x, choices, msg)
```

Arguments

x	Vector of values to test.
choices	Vector to test against.
msg	(character(1)) Error message to display if some elements of x are not elements of choices.

Details

This function is a wrapper for shiny::validate.

Examples

```
ui <- fluidPage(
  selectInput(
    "species",
    "Select species",
    choices = c("setosa", "versicolor", "virginica", "unknown species"),
    selected = "setosa",
    multiple = FALSE
  ),
  verbatimTextOutput("summary")
)

server <- function(input, output) {
  output$summary <- renderPrint({
    validate_in(input$species, iris$Species, "Species does not exist.")
    nrow(iris[iris$Species == input$species, ])
  })
}
if (interactive()) {
  shinyApp(ui, server)
}
```

---

validate_inputs *Send input validation messages to output*

Description

Captures messages from InputValidator objects and collates them into one message passed to validate.

Usage

```
validate_inputs(..., header = "Some inputs require attention")
```

Arguments

...	either any number of InputValidator objects or an optionally named, possibly nested list of InputValidator objects, see Details
header	(character(1)) generic validation message; set to NULL to omit

Details

shiny::validate is used to withhold rendering of an output element until certain conditions are met and to print a validation message in place of the output element. shinyvalidate::InputValidator allows to validate input elements and to display specific messages in their respective input widgets. validate_inputs provides a hybrid solution. Given an InputValidator object, messages corresponding to inputs that fail validation are extracted and placed in one validation message that is passed to a validate/need call. This way the input validator messages are repeated in the output.

The ... argument accepts any number of InputValidator objects or a nested list of such objects. If validators are passed directly, all their messages are printed together under one (optional) header message specified by header. If a list is passed, messages are grouped by validator. The list's names are used as headers for their respective message groups. If neither of the nested list elements is named, a header message is taken from header.

Value

Returns NULL if the final validation call passes and a shiny.silent.error if it fails.

See Also

shinyvalidate::InputValidator, shiny::validate

Examples

```
library(shiny)
library(shinyvalidate)

ui <- fluidPage(
  selectInput("method", "validation method", c("sequential", "combined", "grouped")),
  sidebarLayout(
    sidebarPanel(
      selectInput("letter", "select a letter:", c(letters[1:3], LETTERS[4:6])),
      selectInput("number", "select a number:", 1:6),
      br(),
      selectInput("color", "select a color:",
        c("black", "indianred2", "springgreen2", "cornflowerblue"),
        multiple = TRUE
      ),
      sliderInput("size", "select point size:",
        min = 0.1, max = 4, value = 0.25
      )
    ),
    mainPanel(plotOutput("plot"))
  )
)
```

```
server <- function(input, output) {
  # set up input validation
  iv <- InputValidator$new()
  iv$add_rule("letter", sv_in_set(LETTERS, "choose a capital letter"))
  iv$add_rule("number", function(x) {
    if (as.integer(x) %% 2L == 1L) "choose an even number"
  })
  iv$enable()
  # more input validation
  iv_par <- InputValidator$new()
  iv_par$add_rule("color", sv_required(message = "choose a color"))
  iv_par$add_rule("color", function(x) {
    if (length(x) > 1L) "choose only one color"
  })
  iv_par$add_rule(
    "size",
    sv_between(
      left = 0.5, right = 3,
      message_fmt = "choose a value between {left} and {right}"
    )
  )
  iv_par$enable()

  output$plot <- renderPlot({
    # validate output
    switch(input[["method"]],
      "sequential" = {
        validate_inputs(iv)
        validate_inputs(iv_par, header = "Set proper graphical parameters")
      },
      "combined" = validate_inputs(iv, iv_par),
      "grouped" = validate_inputs(list(
        "Some inputs require attention" = iv,
        "Set proper graphical parameters" = iv_par
      ))
    )

    plot(faithful$eruptions ~ faithful$waiting,
      las = 1, pch = 16,
      col = input[["color"]], cex = input[["size"]]
    )
  })
}

if (interactive()) {
  shinyApp(ui, server)
}
```

validate_no_intersection *Validates no intersection between two vectors*

Description

[Stable]

Usage

```
validate_no_intersection(x, y, msg)
```

Arguments

x	vector
y	vector
msg	(character(1)) message to display if x and y intersect

Details

This function is a wrapper for shiny::validate.

Examples

```
data <- data.frame(
  id = c(1:10, 11:20, 1:10),
  strata = rep(c("A", "B", "C"), each = 10)
)

ui <- fluidPage(
  selectInput("ref1", "Select strata1 to compare",
    choices = c("A", "B", "C"),
    selected = "A"
  ),
  selectInput("ref2", "Select strata2 to compare",
    choices = c("A", "B", "C"),
    selected = "B"
  ),
  verbatimTextOutput("summary")
)

server <- function(input, output) {
  output$summary <- renderText({
    sample_1 <- data$id[data$strata == input$ref1]
    sample_2 <- data$id[data$strata == input$ref2]

    validate_no_intersection(
      sample_1, sample_2,
      "subjects within strata1 and strata2 cannot overlap"
    )
    paste0(
```

```
      "Number of subject in: reference treatment=", length(sample_1),
      " comparions treatment=", length(sample_2)
    )
  })
}
if (interactive()) {
  shinyApp(ui, server)
}
```

`validate_n_levels` *Validate that variables has expected number of levels*

Description

[Stable]

Usage

```
validate_n_levels(x, min_levels = 1, max_levels = 12, var_name)
```

Arguments

`x`	variable name. If `x` is not a factor, the unique values are treated as levels.
`min_levels`	cutoff for minimum number of levels of `x`
`max_levels`	cutoff for maximum number of levels of `x`
`var_name`	name of variable being validated for use in validation message

Details

If the number of levels of `x` is less than `min_levels` or greater than `max_levels` the validation will fail. This function is a wrapper for `shiny::validate`.

Examples

```
data <- data.frame(
  one = rep("a", length.out = 20),
  two = rep(c("a", "b"), length.out = 20),
  three = rep(c("a", "b", "c"), length.out = 20),
  four = rep(c("a", "b", "c", "d"), length.out = 20),
  stringsAsFactors = TRUE
)
ui <- fluidPage(
  selectInput(
    "var",
    "Select variable",
    choices = c("one", "two", "three", "four"),
    selected = "one"
```

```
  ),
  verbatimTextOutput("summary")
)

server <- function(input, output) {
  output$summary <- renderText({
   validate_n_levels(data[[input$var]], min_levels = 2, max_levels = 15, var_name = input$var)
    paste0(
      "Levels of selected treatment variable: ",
      paste(levels(data[[input$var]]),
        collapse = ", "
      )
    )
  })
}
if (interactive()) {
  shinyApp(ui, server)
}
```

`validate_one_row_per_id` *Validate that dataset has unique rows for key variables*

Description

[Stable]

Usage

```
validate_one_row_per_id(x, key = c("USUBJID", "STUDYID"))
```

Arguments

x	(`data.frame`)
key	(`character`) Vector of ID variables from `x` that identify unique records.

Details

This function is a wrapper for `shiny::validate`.

Examples

```
iris$id <- rep(1:50, times = 3)
ui <- fluidPage(
  selectInput(
    inputId = "species",
    label = "Select species",
    choices = c("setosa", "versicolor", "virginica"),
    selected = "setosa",
```

```
    multiple = TRUE
  ),
  plotOutput("plot")
)
server <- function(input, output) {
  output$plot <- renderPlot({
    iris_f <- iris[iris$Species %in% input$species, ]
    validate_one_row_per_id(iris_f, key = c("id"))

    hist(iris_f$Sepal.Length, breaks = 5)
  })
}
if (interactive()) {
  shinyApp(ui, server)
}
```

Index