Package: teal (via r-universe)

October 17, 2024

Type Package

Title Exploratory Web Apps for Analyzing Clinical Trials Data

Version 0.15.2.9072

Date 2024-10-17

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 (>= 1.8.1), teal.data (> 0.6.0.9007), teal.slice (>= 0.5.1.9009)

Imports checkmate (>= 2.1.0), jsonlite, lifecycle (>= 0.2.0), logger (>= 0.2.0), methods, rlang (>= 1.0.0), shinyjs, stats, teal.code (>= 0.5.0), teal.logger (>= 0.2.0), teal.reporter (>= 0.3.1.9004), teal.widgets (>= 0.4.0), utils

Suggests bslib, knitr (>= 1.42), mirai (>= 1.1.1), MultiAssayExperiment, R6, renv (>= 1.0.10.9000), rmarkdown (>= 2.23), rvest (>= 1.0.0), shinytest2, shinyvalidate, testthat (>= 3.2.0), withr (>= 2.1.0), yaml (>= 1.1.0)

VignetteBuilder knitr

RdMacros lifecycle

Remotes rstudio/renv

Config/Needs/verdepcheck rstudio/shiny, insightsengineering/teal.data, insightsengineering/teal.slice, mllg/checkmate, jeroen/jsonlite, r-lib/lifecycle, daroczig/logger, shikokuchuo/mirai, shikokuchuo/nanonext, rstudio/renv, r-lib/rlang, daattali/shinyjs, insightsengineering/teal.code, insightsengineering/teal.logger,

insightsengineering/teal.reporter, insightsengineering/teal.widgets, rstudio/bslib, yihui/knitr, bioc::MultiAssayExperiment, r-lib/R6, rstudio/rmarkdown, tidyverse/rvest, rstudio/shinytest2, rstudio/shinyvalidate, 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 7.3.2

Collate 'TealAppDriver.R' 'checkmate.R' 'dummy_functions.R' 'get_rcode_utils.R' 'include_css_js.R' 'modules.R' 'init.R' 'landing_popup_module.R' 'module_bookmark_manager.R' 'module_data_summary.R' 'module_filter_data.R' 'module_filter_manager.R' 'module_init_data.R' 'module_nested_tabs.R' 'module_snapshot_manager.R' 'module_teal.R' 'module_teal_data.R' 'module_teal_lockfile.R' 'module_teal_with_splash.R' 'module_transform_data.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_data_utils.R' 'teal_reporter.R' 'teal_slices-store.R' 'teal_slices.R' 'utils.R' 'validate_inputs.R' 'validations.R' 'zzz.R'

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

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

RemoteRef HEAD

RemoteSha 89e80289e420996e1ee6931f4de63130606eecf2

Contents

`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",
  transformers = list()
)
```

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) Names of the datasets relevant to the item. There are 2 reserved values that have specific behaviors: • The keyword "all" includes all datasets available in the data passed to the teal application. • NULL hides the sidebar panel completely. • If transformers are specified, their datanames are automatically added to this datanames argument.
transformers	(list of teal_data_module) that will be applied to transform the data. Each transform module UI will appear in the teal's sidebar panel. Transformers' datanames are added to the datanames. See teal_transform_module().

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)
}
```

`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),
  landing_popup = NULL
)
```

Arguments

`data`	(`teal_data` or `teal_data_module`) For constructing the data object, refer to `teal_data()` and `teal_data_module()`. If `datanames` are not set for the `teal_data` object, defaults from the `teal_data` environment will be used.
`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`) Optionally, specifies the initial filter using `teal_slices()`.
`title`	(`shiny.tag` or `character(1)`) Optionally, 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)`) Optionally, the header of the app.
`footer`	(`shiny.tag` or `character(1)`) Optionally, the footer of the app.
`id`	(`character`) Optionally, a 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.
`landing_popup`	(`teal_module_landing`) Optionally, a `landing_popup_module` to show up as soon as the teal app is initialized.

Value

Named list containing server and UI functions.

Examples

```
app <- init(
  data = within(
    teal_data(),
    {
      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, ...) tags$div(p("information about data source")),
      datanames = "all"
    ),
    example_module(label = "example teal module"),
    module(
      "Iris Sepal.Length histogram",
      server = function(input, output, session, data) {
        output$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$p("Sample footer")
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

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

```
app2 <- init(
  data = teal_data(iris = iris),
  modules = modules(
    example_module()
  ),
  landing_popup = 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()")
    )
  )
)

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

`module_teal` `teal` *main module*

Description

[Stable] Module to create a `teal` app. This module can be called directly instead of `init()` and included in your custom application. Please note that `init()` adds `reporter_previewer_module` automatically, which is not a case when calling `ui/srv_teal` directly.

Usage

```
ui_teal(
  id,
  modules,
  title = build_app_title(),
  header = tags$p(),
  footer = tags$p()
)

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

Arguments

id	(character) Optionally, a 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.
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.
title	(shiny.tag or character(1)) Optionally, 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)) Optionally, the header of the app.
footer	(shiny.tag or character(1)) Optionally, the footer of the app.
data	(teal_data, teal_data_module, or reactive returning teal_data) The data which application will depend on.
filter	(teal_slices) Optionally, specifies the initial filter using teal_slices().

Details

Module is responsible for creating the main shiny app layout and initializing all the necessary components. This module establishes reactive connection between the input data and every other component in the app. Reactive change of the data passed as an argument, reloads the app and possibly keeps all input settings the same so the user can continue where one left off.

data flow in teal application:

This module supports multiple data inputs but eventually, they are all converted to reactive returning teal_data in this module. On this reactive teal_data object several actions are performed:

- data loading in module_init_data
- data filtering in module_filter_data
- data transformation in module_transform_data

Fallback on failure:

teal is designed in such way that app will never crash if the error is introduced in any custom shiny module provided by app developer (e.g. teal_data_module(), teal_transform_module()). If any module returns a failing object, the app will halt the evaluation and display a warning message. App user should always have a chance to fix the improper input and continue without restarting the session.

Value

NULL invisibly

module_teal_with_splash *UI and server modules of* teal

Description

[Deprecated] Please use module_teal instead.

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) Optionally, a 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.
data	(teal_data, teal_data_module, or reactive returning teal_data) The data which application will depend on.
title	(shiny.tag or character(1)) Optionally, 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)) Optionally, the header of the app.
footer	(shiny.tag or character(1)) Optionally, the footer of the app.
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) Optionally, specifies the initial filter using teal_slices().

Value

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

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

[Deprecated]

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, defaults to `shiny::getDefaultReactiveDomain()`.

References

`shiny::showModal()`

tdata *Create a* tdata *object*

Description

[Superseded]

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.

Usage

```
new_tdata(...)

tdata2env(...)

get_code_tdata(...)

## S3 method for class 'tdata'
join_keys(...)

get_metadata(...)

as_tdata(...)
```

Arguments

...	ignored

Value

nothing

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, label = "data module", once = TRUE)

## 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
label	(character(1)) Label of the module.
once	(logical(1)) If TRUE, the data module will be shown only once and will disappear after successful data loading. App user will no longer be able to interact with this module anymore. If FALSE, the data module can be reused multiple times. App user will be able to interact and change the data output from the module multiple times.
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 interactively supply or modify data in a teal application. The module allows for running any code (creation *and* some modification) after the app starts or reloads. 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 a list of class teal_data_module containing two elements, ui and server provided via arguments.

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, data, ...) moduleServer(id, function(input, output, session)
    NULL),
  ui = function(id, ...) tags$p(paste0("This module has no UI (id: ", id, " )")),
  filters,
  datanames = "all",
  server_args = NULL,
```

```
  ui_args = NULL,
  transformers = list()
)

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, ...)

set_datanames(modules, datanames)

## 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) Names of the datasets relevant to the item. There are 2 reserved values that have specific behaviors: • The keyword "all" includes all datasets available in the data passed to the teal application. • NULL hides the sidebar panel completely. • If transformers are specified, their datanames are automatically added to this datanames argument.
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.
transformers	(list of teal_data_module) that will be applied to transform the data. Each transform module UI will appear in the teal's sidebar panel. Transformers' datanames are added to the datanames. See teal_transform_module().
...	• 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.
modules	(teal_module or teal_modules)

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.

Restricting datasets used by teal_module:

The datanames argument controls which datasets are used by the module's server. These datasets, passed via server's data argument, are the only ones shown in the module's tab.

When datanames is set to "all", all datasets in the data object are treated as relevant. However, this may include unnecessary datasets, such as:

- Proxy variables for column modifications
- Temporary datasets used to create final versions
- Connection objects

To exclude irrelevant datasets, use the `set_datanames()` function to change `datanames` from `"all"` to specific names. Trying to modify non-`"all"` values with `set_datanames()` will result in a warning. Datasets with names starting with `.` are ignored globally unless explicitly listed in `datanames`.

`datanames` with `transformers`

When transformers are specified, their `datanames` are added to the module's `datanames`, which changes the behavior as follows:

- If `module(datanames)` is `NULL` and the `transformers` have defined `datanames`, the sidebar will appear showing the `transformers`' datasets, instead of being hidden.
- If `module(datanames)` is set to specific values and any `transformer` has `datanames = "all"`, the module may receive extra datasets that could be unnecessary

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)
}
# change the module's datanames
set_datanames(module(datanames = "all"), "a")

# change modules' datanames
set_datanames(
  modules(
    module(datanames = "all"),
    module(datanames = "a")
  ),
  "b"
)
```

`teal_transform_module` *Data module for* `teal` *transformers.*

Description

[Experimental]

Create a `teal_data_module` object for custom transformation of data for pre-processing before passing the data into the module.

Usage

```
teal_transform_module(
  ui = function(id) NULL,
  server = function(id, data) data,
```

```
  label = "transform module",
  datanames = character(0)
)
```

Arguments

ui	(function(id)) shiny module UI function; must only take id argument
server	(function(id, data)) shiny module server function; that takes id and data argument, where the id is the module id and data is the reactive teal_data input. The server function must return reactive expression containing teal_data object. The server function definition should not use eventReactive as it may lead to unexpected behavior. See vignettes("data-transform-as-shiny-module") for more information.
label	(character(1)) Label of the module.
datanames	(character) Names of the datasets that are relevant for this module to evaluate. If set to character(0) then module would receive modules() datanames.

Details

teal_transform_module creates a teal_data_module object to transform data in a teal application. This transformation happens after the data has passed through the filtering activity in teal. The transformed data is then sent to the server of the teal_module().

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

Examples

```
my_transformers <- list(
  teal_transform_module(
    label = "Custom transform for iris",
    datanames = "iris",
    ui = function(id) {
      ns <- NS(id)
      tags$div(
       numericInput(ns("n_rows"), "Subset n rows", value = 6, min = 1, max = 150, step = 1)
      )
    },
    server = function(id, data) {
      moduleServer(id, function(input, output, session) {
        reactive({
          within(data(),
            {
              iris <- head(iris, num_rows)
            },
            num_rows = input$n_rows
          )
        })
      })
    }
```

```
  )
)
```

---

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),
      tags$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