Package 'teal'

May 8, 2026

Type Package

Title Exploratory Web Apps for Analyzing Clinical Trials Data

Version 1.1.0

Date 2025-11-17

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 (>= 4.1), shiny (>= 1.8.1), teal.data (>= 0.8.0), teal.slice (>= 0.7.0)

Imports bsicons, bslib (>= 0.8.0), checkmate (>= 2.1.0), cli, htmltools, jsonlite, lifecycle (>= 0.2.0), logger (>= 0.2.0), methods, rlang (>= 1.0.0), shinyjs, stats, teal.code (>= 0.7.0), teal.logger (>= 0.4.0), teal.reporter (>= 0.5.0.9001), teal.widgets (>= 0.5.0), tools, utils

Suggests ggplot2 (>= 3.4.0), knitr (>= 1.42), mirai (>= 1.1.1), MultiAssayExperiment, R6, renv (>= 1.0.11), rmarkdown (>= 2.23), roxy.shinylive, rvest (>= 1.0.0), shinytest2, shinyvalidate, testthat (>= 3.2.0), withr (>= 2.1.0), yaml (>= 1.1.0)

VignetteBuilder knitr, rmarkdown

RdMacros lifecycle

Config/Needs/verdepcheck rstudio/shiny, insightsengineering/teal.data, insightsengineering/teal.slice, mllg/checkmate, jeroen/jsonlite, r-lib/lifecycle, daroczig/logger, r-lib/mirai, r-lib/cli, r-lib/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,
rstudio/htmltools, bioc::matrixStats,
insightsengineering/roxy.shinylive

Config/Needs/website insightsengineering/nesttemplate

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

Collate 'TealAppDriver.R' 'after.R' 'checkmate.R' 'dummy_functions.R'
'include_css_js.R' 'modules.R' 'init.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_session_info.R' 'module_snapshot_manager.R'
'module_source_code.R' 'module_teal.R' 'module_teal_lockfile.R'
'module_teal_reporter.R' 'module_transform_data.R'
'module_validate_error.R' 'reporter_previewer_module.R'
'teal.R' 'teal_data_module.R' 'teal_data_module-eval_code.R'
'teal_data_module-within.R' 'teal_data_utils.R'
'teal_modifiers.R' 'teal_slices-store.R' 'teal_slices.R'
'teal_transform_module.R' 'utils.R' 'validate_inputs.R'
'validations.R' 'zzz.R'

NeedsCompilation no

Author Dawid Kaledkowski [aut, cre] (ORCID:
<https://orcid.org/0000-0001-9533-457X>),
Pawel Rucki [aut],
Aleksander Chlebowski [aut] (ORCID:
<https://orcid.org/0000-0001-5018-6294>),
Andre Verissimo [aut] (ORCID: <https://orcid.org/0000-0002-2212-339X>),
Kartikeya Kirar [aut],
Vedha Viyash [aut],
Marcin Kosinski [aut],
Adrian Waddell [aut],
Dony Unardi [rev],
Nina Qi [rev],
Nikolas Burkoff [aut],
Mahmoud Hallal [aut],
Maciej Nasinski [aut],
Konrad Pagacz [aut],
Junlue Zhao [aut],
Tadeusz Lewandowski [aut],
Chendi Liao [rev],

F. Hoffmann-La Roche AG [cph, fnd],
Maximilian Mordig [ctb]

Maintainer Dawid Kaledkowski <dawid.kaledkowski@roche.com>

Repository CRAN

Date/Publication 2025-11-17 12:40:02 UTC

Contents

`add_landing_modal` *Add a Landing Popup to* `teal` *Application*

Description

Adds a landing popup to the `teal` app. This popup will be shown when the app starts. The dialog must be closed by the app user to proceed to the main application.

Usage

```
add_landing_modal(
  x,
  title = NULL,
  content = NULL,
  footer = modalButton("Accept"),
  ...
)
```

Arguments

x	(teal_app) A teal_app object created using the init function.
title	An optional title for the dialog.
content	(character(1), shiny.tag or shiny.tag.list) with the content of the popup.
footer	UI for footer. Use NULL for no footer.
...	Additional arguments to shiny::modalDialog().

Examples in Shinylive

example-1 Open in Shinylive

Examples

```
app <- init(
  data = teal_data(IRIS = iris, MTCARS = mtcars),
  modules = modules(example_module())
) |>
  add_landing_modal(
    title = "Welcome",
    content = "This is a landing popup.",
    buttons = modalButton("Accept")
  )

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

---

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.

---

disable_report *Disable the report for a* teal_module

Description

Convenience function that disables the user's ability to add the module to the report previewer.

Usage

```
disable_report(x)
```

Arguments

x	(teal_module) a teal_module object.

Value

modified data object that indicates that it should disable the reporter functionality.

Examples in Shinylive

example-1 Open in Shinylive

example-2 Open in Shinylive

example-3 Open in Shinylive

See Also

disable_src()

Examples

```
# Disabling report on a single module
app <- init(
  data = within(teal_data(), iris <- iris),
  modules = modules(
    example_module(label = "example teal module") |> disable_report()
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# Disabling report on multiple modules
app <- init(
  data = within(teal_data(), iris <- iris),
  modules = modules(
    example_module(label = "example 1"),
    example_module(label = "example 2")
  ) |> disable_report()
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
# Disabling reporting for the app
app <- init(
  data = within(teal_data(), iris <- iris),
  modules = modules(
    example_module(label = "example teal module")
  ),
  reporter = NULL
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

---

disable_src *Disable the "Show R Code" global button in the UI*

Description

Convenience function that disables the user's ability to see the code of the module.

Usage

```
disable_src(x)
```

Arguments

x	(teal_module) a teal_module object.

Details

This is equivalent to setting the attribute `teal.enable_src` to `FALSE` on the data object returned by the module.

Value

modified data object that indicates that it should not show the "Show R Code" button in the UI.

Examples in Shinylive

example-1 Open in Shinylive

example-2 Open in Shinylive

See Also

`disable_report()`

Examples

```
# Disabling source on a single module
app <- init(
  data = within(teal_data(), iris <- iris),
  modules = modules(
    example_module(label = "example teal module") |> disable_src()
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
# Multiple modules
app <- init(
  data = within(teal_data(), iris <- iris),
  modules = modules(
    example_module(label = "example 1"),
    example_module(label = "example 2")
  ) |> disable_src()
)

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

`example_module` *An example* `teal` *module*

Description

This module creates an object called `object` that can be modified with decorators. The `object` is determined by what's selected in `Choose a dataset` input in UI. The object can be anything that can be handled by `renderPrint()`. See the `vignette("transform-module-output", package = "teal")` or `teal_transform_module` to read more about decorators.

Usage

```
example_module(
  label = "example teal module",
  datanames = "all",
  transformators = list(),
  decorators = 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 `transformators` are specified, their `datanames` are automatically added to this `datanames` argument.
`transformators`	(`list` of `teal_transform_module`) that will be applied to transform module's data input. To learn more check `vignette("transform-input-data", package = "teal")`.
`decorators`	**[Experimental]** (`list` of `teal_transform_module`) optional, decorator for `object` included in the module.

Value

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

Reporting

This module returns an object of class `teal_module`, that contains a `server` function. Since the server function returns a `teal_report` object, this makes this module reportable, which means that the reporting functionality will be turned on automatically by the `teal` framework.

For more information on reporting in `teal`, see the vignettes:

- `vignette("reportable-shiny-application", package = "teal.reporter")`
- `vignette("adding-support-for-reporting-to-custom-modules", package = "teal")`

Examples in Shinylive

example-1 Open in Shinylive

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

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 = lifecycle::deprecated(),
  header = lifecycle::deprecated(),
  footer = lifecycle::deprecated(),
  id = lifecycle::deprecated(),
  reporter = teal.reporter::Reporter$new()
)
```

Arguments

`data`	(`teal_data` or `teal_data_module`) For constructing the data object, refer to `teal.data::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`) Optionally, specifies the initial filter using `teal_slices()`.
`title`	(`shiny.tag` or `character(1)`) **[Deprecated]** 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. This parameter is no longer supported. Use `modify_title()` on the teal app object instead.

header	(shiny.tag or character(1)) **[Deprecated]** Optionally, the header of the app. This parameter is no longer supported. Use modify_header() on the teal app object instead.
footer	(shiny.tag or character(1)) **[Deprecated]** Optionally, the footer of the app. This parameter is no longer supported. Use modify_footer() on the teal app object instead.
id	**[Deprecated]** (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 parameter is no longer supported. Use ui_teal() and srv_teal() instead.
reporter	(Reporter) object used to store report contents. Set to NULL to globally disable reporting.

Value

Named list containing server and UI functions.

Examples in Shinylive

example-1 Open in Shinylive

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

`make_teal_transform_server` *Make teal_transform_module's server*

Description

A factory function to simplify creation of a `teal_transform_module`'s server. Specified `expr` is wrapped in a shiny module function and output can be passed to the `server` argument in `teal_transform_module()` call. Such a server function can be linked with ui and values from the inputs can be used in the expression. Object names specified in the expression will be substituted with the value of the respective input (matched by the name) - for example in `expression(graph <- graph + ggtitle(title))` object `title` will be replaced with the value of `input$title`.

Usage

```
make_teal_transform_server(expr)
```

Arguments

`expr`	(`language`) An R call which will be evaluated within `teal.data::teal_data` environment.

Value

`function(id, data)` returning `shiny` module

Examples in Shinylive

example-1 Open in Shinylive

Examples

```
trim_iris <- teal_transform_module(
  label = "Simplified interactive transformator for iris",
  datanames = "iris",
  ui = function(id) {
    ns <- NS(id)
    numericInput(ns("n_rows"), "Subset n rows", value = 6, min = 1, max = 150, step = 1)
  },
  server = make_teal_transform_server(expression(iris <- head(iris, n_rows)))
)

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

---

module_session_info `teal` *user session info module*

Description

Module to display the user session info popup and to download a lockfile. Module is included when running `init()` but skipped when using `module_teal`. Please be aware that session info contains R session information, so multiple module's calls will share the same information.

Usage

```
ui_session_info(id)

srv_session_info(id)
```

Arguments

`id`	(`character(1)`) `shiny` module instance id.

Value

`NULL` invisibly

Examples in Shinylive

example-1 Open in Shinylive

Examples

```
ui <- fluidPage(
  ui_session_info("session_info")
)

server <- function(input, output, session) {
  srv_session_info("session_info")
}

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

`module_teal` `teal` *main module*

Description

Module to create a `teal` app as a Shiny Module.

Usage

```
ui_teal(id, modules)

srv_teal(
  id,
  data,
  modules,
  filter = teal_slices(),
  reporter = teal.reporter::Reporter$new()
)
```

Arguments

`id`	(`character(1)`) `shiny` module instance id.
`modules`	(`teal_modules`) `teal_modules` object. These are the specific output modules which will be displayed in the `teal` application. See `modules()` and `module()` for more details.
`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()`.
`reporter`	(`Reporter`) object used to store report contents. Set to `NULL` to globally disable reporting.

Details

This module can be used instead of `init()` in custom Shiny applications. Unlike `init()`, it doesn't automatically include `module_session_info`.

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_transform_data` *Module to transform* `reactive teal_data`

Description

Module calls `teal_transform_module()` in sequence so that `reactive teal_data` output from one module is handed over to the following module's input.

Usage

```
ui_transform_teal_data(id, transformators, class = "well")

srv_transform_teal_data(
  id,
  data,
  transformators,
  modules = NULL,
  is_transform_failed = reactiveValues()
)
```

Arguments

id	(character(1)) shiny module instance id.
transformators	(list of teal_transform_module) that will be applied to transform module's data input. To learn more check vignette("transform-input-data", package = "teal").
class	(character(1)) CSS class to be added in the div wrapper tag.
data	(teal_data, teal_data_module, or reactive returning teal_data) The data which application will depend on.
modules	(teal_modules or teal_module) For datanames validation purpose
is_transform_failed	(reactiveValues) contains logical flags named after each transformator. Help to determine if any previous transformator failed, so that following transformators can be disabled and display a generic failure message.

Value

reactive teal_data

---

reporter_previewer_module

Create a teal *module for previewing a report*

Description

[Deprecated]

This function controls the appearance of the drop-down menu for the reporter. It is now deprecated in favor of the options:

- teal.reporter.nav_buttons = c("preview", "download", "load", "reset") to control which buttons will be displayed in the drop-down.
- teal.reporter.rmd_output: passed to teal.reporter::download_report_button_srv()
- teal.reporter.rmd_yaml_args: passed to teal.reporter::download_report_button_srv()
- teal.reporter.global_knitr: passed to teal.reporter::download_report_button_srv()

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 will overwrite the default teal.reporter options described in the description.

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.

TealReportCard TealReportCard

Description

[Experimental] Child class of `teal.reporter::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]]
```

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]]
```

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]]

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

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

`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'
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, language or expression) code to evaluate. It is possible to preserve original formatting of the code by providing a character or an expression being a result of parse(keep.source = TRUE).
data	(teal_data_module) object
expr	(expression) to evaluate. Must be inline code. See within()
...	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
          }
        )

        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

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,
  transformators = list()
)

modules(..., label = character(0))

## S3 method for class 'teal_module'
format(
  x,
  is_last = FALSE,
  parent_prefix = "",
  what = c("datasets", "properties", "ui_args", "server_args", "decorators",
    "transformators"),
  ...
)

## S3 method for class 'teal_modules'
format(x, is_root = TRUE, is_last = FALSE, parent_prefix = "", ...)

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

## 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) If the server function includes a `data` argument, it will receive a reactive expression containing the `teal_data` object.
- `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 `transformators` 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.

`transformators` (`list` of `teal_transform_module`) that will be applied to transform module's data input. To learn more check `vignette("transform-input-data", package = "teal")`.

`...`

- 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.

`is_last` (`logical(1)`) Whether this is the last item in its parent's children list. Affects the tree branch character used (L- vs |-)

`parent_prefix` (`character(1)`) The prefix inherited from parent nodes, used to maintain the tree structure in nested levels

`what` (`character`) Specifies which metadata to display. Possible values: "datasets", "properties", "ui_args", "server_args", "transformators"

`is_root` (`logical(1)`) Whether this is the root node of the tree. Only used in format.teal_modules(). Determines whether to show "TEAL ROOT" header

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 an object of class teal_modules.

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 ones
- Connection objects

Datasets which name is prefixed in teal_data by the dot (.) are not displayed in the teal application. Please see the *"Hidden datasets"* section in 'vignette("including-data-in-teal-applications").

datanames with transformators

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

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

Examples in Shinylive

example-1 Open in Shinylive

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)
}
mod <- module(
  label = "My Custom Module",
  server = function(id, data, ...) {},
  ui = function(id, ...) {},
  datanames = c("ADSL", "ADTTE"),
  transformators = list(),
  ui_args = list(a = 1, b = "b"),
  server_args = list(x = 5, y = list(p = 1))
)
cat(format(mod))
custom_module <- function(
  label = "label", ui_args = NULL, server_args = NULL,
  datanames = "all", transformators = list(), bk = FALSE
) {
  ans <- module(
    label,
    server = function(id, data, ...) {},
    ui = function(id, ...) {},
    datanames = datanames,
    transformators = transformators,
    ui_args = ui_args,
    server_args = server_args
  )
  attr(ans, "teal_bookmarkable") <- bk
  ans
}

dummy_transformator <- teal_transform_module(
  label = "Dummy Transform",
  ui = function(id) div("(does nothing)"),
  server = function(id, data) {
    moduleServer(id, function(input, output, session) data)
  }
)

plot_transformator <- teal_transform_module(
  label = "Plot Settings",
  ui = function(id) div("(does nothing)"),
  server = function(id, data) {
    moduleServer(id, function(input, output, session) data)
  }
)

static_decorator <- teal_transform_module(
  label = "Static decorator",
  server = function(id, data) {
    moduleServer(id, function(input, output, session) {
      reactive({
        req(data())
        within(data(), {
          plot <- plot +
```

```
            ggtitle("This is title") +
            xlab("x axis")
        })
      })
    })
  }
)

complete_modules <- modules(
  custom_module(
    label = "Data Overview",
    datanames = c("ADSL", "ADAE", "ADVS"),
    ui_args = list(
      view_type = "table",
      page_size = 10,
      filters = c("ARM", "SEX", "RACE"),
      decorators = list(static_decorator)
    ),
    server_args = list(
      cache = TRUE,
      debounce = 1000,
      decorators = list(static_decorator)
    ),
    transformators = list(dummy_transformator),
    bk = TRUE
  ),
  modules(
    label = "Nested 1",
    custom_module(
      label = "Interactive Plots",
      datanames = c("ADSL", "ADVS"),
      ui_args = list(
        plot_type = c("scatter", "box", "line"),
        height = 600,
        width = 800,
        color_scheme = "viridis"
      ),
      server_args = list(
        render_type = "svg",
        cache_plots = TRUE
      ),
      transformators = list(dummy_transformator, plot_transformator),
      bk = TRUE
    ),
    modules(
      label = "Nested 2",
      custom_module(
        label = "Summary Statistics",
        datanames = "ADSL",
        ui_args = list(
          stats = c("mean", "median", "sd", "range"),
          grouping = c("ARM", "SEX")
        )
```

```
      ),
      modules(
        label = "Labeled nested modules",
        custom_module(
          label = "Subgroup Analysis",
          datanames = c("ADSL", "ADAE"),
          ui_args = list(
            subgroups = c("AGE", "SEX", "RACE"),
            analysis_type = "stratified"
          ),
          bk = TRUE
        )
      ),
      modules(custom_module(label = "Subgroup Analysis in non-labled modules"))
    )
  ),
  custom_module("Non-nested module")
)

cat(format(complete_modules))
cat(format(complete_modules, what = c("ui_args", "server_args", "transformators")))
cat(format(complete_modules, what = c("decorators", "transformators")))
```

`teal_transform_module` *Data module for* `teal` *transformations and output customization*

Description

[Experimental]

`teal_transform_module` provides a `shiny` module that enables data transformations within a `teal` application and allows for customization of outputs generated by modules.

Usage

```
teal_transform_module(
  ui = NULL,
  server = function(id, data) data,
  label = "transform module",
  datanames = "all"
)
```

Arguments

`ui`	(`function(id)`) `shiny` module UI function; must only take `id` argument
`server`	(`function(id, data)` or `expression`) A `shiny` module server function that takes `id` and `data` as arguments, where `id` is the module id and `data` is the reactive `teal_data` input. The `server` function must return a reactive expression containing a `teal_data` object. For simplified syntax, use `make_teal_transform_server()`.

label	(character(1)) Label of the module.
datanames	(character) Specifies the names of datasets relevant to the module. Only filters for the specified datanames will be displayed in the filter panel. The keyword "all" can be used to display filters for all datasets. datanames are automatically appended to the modules() datanames.

Transforming Module Inputs in teal

Data transformations occur after data has been filtered in teal. The transformed data is then passed to the server of teal_module() and managed by teal's internal processes. The primary advantage of teal_transform_module over custom modules is in its error handling, where all warnings and errors are managed by teal, allowing developers to focus on transformation logic.

For more details, see the vignette: vignette("transform-input-data", package = "teal").

Customizing Module Outputs

teal_transform_module also allows developers to modify any object created within teal.data::teal_data. This means you can use it to customize not only datasets but also tables, listings, and graphs. Some teal_modules permit developers to inject custom shiny modules to enhance displayed outputs. To manage these decorators within your module, use ui_transform_teal_data() and srv_transform_teal_data(). (For further guidance on managing decorators, refer to ui_args and srv_args in the vignette documentation.)

See the vignette vignette("transform-module-output", package = "teal") for additional examples.

server as a language

The server function in teal_transform_module must return a reactive teal.data::teal_data object. For simple transformations without complex reactivity, the server function might look like this:s

```
function(id, data) {
  moduleServer(id, function(input, output, session) {
    reactive({
      within(
        data(),
        expr = x <- subset(x, col == level),
        level = input$level
      )
    })
  })
}
```

The example above can be simplified using make_teal_transform_server, where level is automatically matched to the corresponding input parameter:

```
make_teal_transform_server(expr = expression(x <- subset(x, col == level)))
```

Examples in Shinylive

example-1 Open in Shinylive

Examples

```
data_transformators <- list(
  teal_transform_module(
    label = "Static transformator for iris",
    datanames = "iris",
    server = function(id, data) {
      moduleServer(id, function(input, output, session) {
        reactive({
          within(data(), {
            iris <- head(iris, 5)
          })
        })
      })
    }
  ),
  teal_transform_module(
    label = "Interactive transformator for iris",
    datanames = "iris",
    ui = function(id) {
      ns <- NS(id)
      tags$div(
       numericInput(ns("n_cols"), "Show n columns", value = 5, min = 1, max = 5, step = 1)
      )
    },
    server = function(id, data) {
      moduleServer(id, function(input, output, session) {
        reactive({
          within(data(),
            {
              iris <- iris[, 1:n_cols]
            },
            n_cols = input$n_cols
          )
        })
      })
    }
  )
)

output_decorator <- teal_transform_module(
  server = make_teal_transform_server(
    expression(
      object <- rev(object)
    )
  )
)

app <- init(
```

```
  data = teal_data(iris = iris),
  modules = example_module(
    transformators = data_transformators,
    decorators = list(output_decorator)
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

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

Description

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

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.

Examples in Shinylive

example-1 Open in Shinylive

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

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

Usage

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

Arguments

x	vector
msg	message to display

Examples in Shinylive

example-1 Open in Shinylive

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

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

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`

Examples in Shinylive

example-1 Open in Shinylive

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

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

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`.

Examples in Shinylive

example-1 Open in Shinylive

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.

Examples in Shinylive

example-1 Open in Shinylive

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

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

Usage

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

Arguments

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

Examples in Shinylive

example-1 Open in Shinylive

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

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`.

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

Examples in Shinylive

example-1 Open in Shinylive

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

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

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.

Examples in Shinylive

example-1 Open in Shinylive

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