Package 'datamods'

May 8, 2026

Title Modules to Import and Manipulate Data in 'Shiny'

Version 1.5.3

Description 'Shiny' modules to import data into an application or 'addin' from various sources, and to manipulate them after that.

License GPL-3

URL https://github.com/dreamRs/datamods, https://dreamrs.github.io/datamods/

BugReports https://github.com/dreamRs/datamods/issues

Encoding UTF-8

RoxygenNote 7.3.2

Imports bslib, classInt, data.table, htmltools, phosphoricons, reactable, readxl, rio, rlang, shiny (>= 1.5.0), shinyWidgets (>= 0.8.4), tibble, toastui (>= 0.3.3), tools, shinybusy, writexl

Suggests ggplot2, jsonlite, knitr, MASS, rmarkdown, testthat, validate

VignetteBuilder knitr

Depends R (>= 2.10)

LazyData true

NeedsCompilation no

Author Victor Perrier [aut, cre, cph], Fanny Meyer [aut], Samra Goumri [aut], Zauad Shahreer Abeer [aut], Eduard Szöcs [ctb]

Maintainer Victor Perrier <victor.perrier@dreamrs.fr>

Repository CRAN

Date/Publication 2024-10-02 10:40:03 UTC

Contents

`create-column` *Create new column*

Description

This module allow to enter an expression to create a new column in a `data.frame`.

Usage

```
create_column_ui(id)

create_column_server(
  id,
  data_r = reactive(NULL),
  allowed_operations = list_allowed_operations()
)

list_allowed_operations()

modal_create_column(
  id,
  title = i18n("Create a new column"),
  easyClose = TRUE,
```

```
  size = "l",
  footer = NULL
)

winbox_create_column(
  id,
  title = i18n("Create a new column"),
  options = shinyWidgets::wbOptions(),
  controls = shinyWidgets::wbControls()
)

winbox_update_factor(
  id,
  title = i18n("Update levels of a factor"),
  options = shinyWidgets::wbOptions(),
  controls = shinyWidgets::wbControls()
)
```

Arguments

id	Module's ID.
data_r	A shiny::reactive() function returning a data.frame.
allowed_operations	A list of allowed operations, see below for details.
title	An optional title for the dialog.
easyClose	If TRUE, the modal dialog can be dismissed by clicking outside the dialog box, or be pressing the Escape key. If FALSE (the default), the modal dialog can't be dismissed in those ways; instead it must be dismissed by clicking on a modalButton(), or from a call to removeModal() on the server.
size	One of "s" for small, "m" (the default) for medium, "l" for large, or "xl" for extra large. Note that "xl" only works with Bootstrap 4 and above (to opt-in to Bootstrap 4+, pass bslib::bs_theme() to the theme argument of a page container like fluidPage()).
footer	UI for footer. Use NULL for no footer.
options	List of options, see wbOptions().
controls	List of controls, see wbControls().

Value

A shiny::reactive() function returning the data.

Note

User can only use a subset of function: (, c, +, -, *, ^, %%, %/%, /, ==, >, <, !=, <=, >=, &, |, abs, sign, sqrt, ceiling, floor, trunc, cummax, cummin, cumprod, cumsum, exp, expm1, log, log10, log2, log1p, cos, cosh, sin, sinh, tan, tanh, acos, acosh, asin, asinh, atan, atanh, cospi, sinpi, tanpi, gamma, lgamma, digamma, trigamma, round, signif, max, min, range, prod, sum, any, all,

pmin, pmax, mean, paste, paste0, substr, nchar, trimws, gsub, sub, grepl, ifelse, length, as.numeric, as.character, as.integer, as.Date, as.POSIXct, as.factor, factor. You can add more operations using the `allowed_operations` argument, for example if you want to allow to use package lubridate, you can do:

```
c(list_allowed_operations(), getNamespaceExports("lubridate"))
```

Examples

```
library(shiny)
library(datamods)
library(reactable)

ui <- fluidPage(
  theme = bslib::bs_theme(version = 5L, preset = "bootstrap"),
  shinyWidgets::html_dependency_winbox(),
  tags$h2("Create new column"),
  fluidRow(
    column(
      width = 4,
      create_column_ui("inline"),
      actionButton("modal", "Or click here to open a modal to create a column"),
      tags$br(), tags$br(),
      actionButton("winbox", "Or click here to open a WinBox to create a column")
    ),
    column(
      width = 8,
      reactableOutput(outputId = "table"),
      verbatimTextOutput("code")
    )
  )
)

server <- function(input, output, session) {

  rv <- reactiveValues(data = MASS::Cars93[, c(1, 3, 4, 5, 6, 10)])

  # inline mode
  data_inline_r <- create_column_server(
    id = "inline",
    data_r = reactive(rv$data)
  )
  observeEvent(data_inline_r(), rv$data <- data_inline_r())

  # modal window mode
  observeEvent(input$modal, modal_create_column("modal"))
  data_modal_r <- create_column_server(
    id = "modal",
    data_r = reactive(rv$data)
  )
  observeEvent(data_modal_r(), rv$data <- data_modal_r())
```

```
  # WinBox window mode
  observeEvent(input$winbox, winbox_create_column("winbox"))
  data_winbox_r <- create_column_server(
    id = "winbox",
    data_r = reactive(rv$data)
  )
  observeEvent(data_winbox_r(), rv$data <- data_winbox_r())

  # Show result
  output$table <- renderReactable({
    data <- req(rv$data)
    reactable(
      data = data,
      bordered = TRUE,
      compact = TRUE,
      striped = TRUE
    )
  })

  output$code <- renderPrint({
    attr(rv$data, "code")
  })
}

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

---

cut-variable *Module to Convert Numeric to Factor*

Description

This module contain an interface to cut a numeric into several intervals.

Usage

```
cut_variable_ui(id)

cut_variable_server(id, data_r = reactive(NULL))

modal_cut_variable(
  id,
  title = i18n("Convert Numeric to Factor"),
  easyClose = TRUE,
  size = "l",
  footer = NULL
)

winbox_cut_variable(
```

```
  id,
  title = i18n("Convert Numeric to Factor"),
  options = shinyWidgets::wbOptions(),
  controls = shinyWidgets::wbControls()
)
```

Arguments

id	Module ID.
data_r	A shiny::reactive() function returning a data.frame.
title	An optional title for the dialog.
easyClose	If TRUE, the modal dialog can be dismissed by clicking outside the dialog box, or be pressing the Escape key. If FALSE (the default), the modal dialog can't be dismissed in those ways; instead it must be dismissed by clicking on a modalButton(), or from a call to removeModal() on the server.
size	One of "s" for small, "m" (the default) for medium, "l" for large, or "xl" for extra large. Note that "xl" only works with Bootstrap 4 and above (to opt-in to Bootstrap 4+, pass bslib::bs_theme() to the theme argument of a page container like fluidPage()).
footer	UI for footer. Use NULL for no footer.
options	List of options, see wbOptions().
controls	List of controls, see wbControls().

Value

A shiny::reactive() function returning the data.

Examples

```
library(shiny)
library(datamods)
library(reactable)

ui <- fluidPage(
  theme = bslib::bs_theme(version = 5L, preset = "bootstrap"),
  shinyWidgets::html_dependency_winbox(),
  tags$h2("Convert Numeric to Factor"),
  fluidRow(
    column(
      width = 6,
      cut_variable_ui("inline"),
      actionButton("modal", "Or click here to open a modal to cut a variable"),
      tags$br(), tags$br(),
      actionButton("winbox", "Or click here to open a WinBox to cut a variable")
    ),
    column(
      width = 6,
      reactableOutput(outputId = "table"),
      verbatimTextOutput("code")
```

```
    )
  )
)

server <- function(input, output, session) {

  rv <- reactiveValues(data = MASS::Cars93[, c(1, 3, 4, 5, 6, 10)])

  # inline mode
  data_inline_r <- cut_variable_server(
    id = "inline",
    data_r = reactive(rv$data)
  )
  observeEvent(data_inline_r(), rv$data <- data_inline_r())

  # modal window mode
  observeEvent(input$modal, modal_cut_variable("modal"))
  data_modal_r <- cut_variable_server(
    id = "modal",
    data_r = reactive(rv$data)
  )
  observeEvent(data_modal_r(), rv$data <- data_modal_r())

  # WinBox window mode
  observeEvent(input$winbox, winbox_cut_variable("winbox"))
  data_winbox_r <- cut_variable_server(
    id = "winbox",
    data_r = reactive(rv$data)
  )
  observeEvent(data_winbox_r(), rv$data <- data_winbox_r())

  # Show result
  output$table <- renderReactable({
    data <- req(rv$data)
    reactable(
      data = data,
      bordered = TRUE,
      compact = TRUE,
      striped = TRUE
    )
  })

  output$code <- renderPrint({
    attr(rv$data, "code")
  })
}

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

---

demo_edit *Customer Credit Card Information*

Description

A subset of fake customer credit card information inspired by the {charlatan} package.

Usage

```
demo_edit
```

Format

demo_edit:
A data frame with 20 rows and 6 columns:

name Customer name
job Customer job
credit_card_provider Credit card provider
credit_card_security_code Credit card security code
date_obtained Date of obtaining the credit card
contactless_card Contactless card

Source

https://CRAN.R-project.org/package=charlatan

---

edit-data *Shiny module to interactively edit a* data.frame

Description

The module generates different options to edit a data.frame: adding, deleting and modifying rows, exporting data (csv and excel), choosing editable columns, choosing mandatory columns. This module returns the edited table with the user modifications.

Usage

```
edit_data_ui(id)

edit_data_server(
  id,
  data_r = reactive(NULL),
  add = TRUE,
  update = TRUE,
  delete = TRUE,
  download_csv = TRUE,
  download_excel = TRUE,
  file_name_export = "data",
  var_edit = NULL,
  var_mandatory = NULL,
```

```
  var_labels = NULL,
  add_default_values = list(),
  n_column = 1,
  return_class = c("data.frame", "data.table", "tbl_df", "raw"),
  reactable_options = NULL,
  modal_size = c("m", "s", "l", "xl"),
  modal_easy_close = TRUE,
  callback_add = NULL,
  callback_update = NULL,
  callback_delete = NULL,
  only_callback = FALSE,
  use_notify = TRUE
)
```

Arguments

id	Module ID
data_r	data_r reactive function containing a data.frame to use in the module.
add	boolean, if TRUE, allows you to add a row in the table via a button at the top right.
update	boolean, if TRUE, allows you to modify a row of the table via a button located in the table on the row you want to edit.
delete	boolean, if TRUE, allows a row to be deleted from the table via a button in the table.
download_csv	if TRUE, allows to export the table in csv format via a download button.
download_excel	if TRUE, allows to export the table in excel format via a download button.
file_name_export	character that allows you to choose the export name of the downloaded file.
var_edit	vector of character which allows to choose the names of the editable columns.
var_mandatory	vector of character which allows to choose obligatory fields to fill.
var_labels	named list, where names are colnames and values are labels to be used in edit modal.
add_default_values	Default values to use for input control when adding new data, e.g. list(my_var_text = "Default text to display").
n_column	Number of column in the edit modal window, must be a number that divide 12 since it use Bootstrap grid system with shiny::column().
return_class	Class of returned data: data.frame, data.table, tbl_df (tibble) or raw.
reactable_options	Options passed to reactable::reactable().
modal_size	character which allows to choose the size of the modalDialog. One of "s" for small, "m" (the default) for medium, "l" for large, or "xl" for extra large.
modal_easy_close	boolean If TRUE, modalDialog can be dismissed by clicking outside the dialog box, or be pressing the Escape key. If FALSE (the default), modalDialog

can't be dismissed in those ways; instead it must be dismissed by clicking on a modalButton(), or from a call to removeModal() on the server.

`callback_add`, `callback_update`, `callback_delete`

Functions to be executed just before an action (add, update or delete) is performed on the data. Functions used must be like `function(data, row) {...}` where :

- `data` will be the data in the table at the moment the function is called
- `row` will contain either a new row of data (add), an updated row (update) or the row that will be deleted (delete).

If the return value of a callback function is not truthy (see `shiny::isTruthy()`) then the action is cancelled.

`only_callback` Only use callbacks, don't alter data within the module.

`use_notify` Display information or not to user through `shinybusy::notify()`.

Value

the edited `data.frame` in reactable format with the user modifications

Examples

```
library(shiny)
library(datamods)
library(bslib)
library(reactable)

ui <- fluidPage(
  theme = bs_theme(
    version = 5
  ),
  tags$h2("Edit data", align = "center"),
  edit_data_ui(id = "id"),
  verbatimTextOutput("result")
)


server <- function(input, output, session) {

  edited_r <- edit_data_server(
    id = "id",
    data_r = reactive(demo_edit),
    add = TRUE,
    update = TRUE,
    delete = TRUE,
    download_csv = TRUE,
    download_excel = TRUE,
    file_name_export = "datas",
    # var_edit = c("name", "job", "credit_card_provider", "credit_card_security_code"),
    var_mandatory = c("name", "job"),
    var_labels = list(
      name = "Name",
```

```
    credit_card_security_code = "Credit card security code",
    date_obtained = "Date obtained",
    contactless_card = "Contactless Card",
    credit_card_provider = "Credit card provider"
  ),
  add_default_values = list(
    name = "Please enter your name here",
    date_obtained = Sys.Date()
  ),
  n_column = 2,
  modal_size = "l",
  modal_easy_close = TRUE,
  reactable_options = list(
    defaultColDef = colDef(filterable = TRUE),
    selection = "single",
    columns = list(
      name = colDef(name = "Name", style = list(fontWeight = "bold")),
      credit_card_security_code = colDef(name = "Credit card security code"),
      date_obtained = colDef(name = "Date obtained", format = colFormat(date = TRUE)),
      contactless_card = colDef(
        name = "Contactless Card",
        cell = function(value) {
          # Render as an X mark or check mark
          if (value == FALSE) "\u274c No" else "\u2714\ufe0f Yes"
        }),
      credit_card_provider = colDef(
        name = "Credit card provider",
        style = function(value) {
          if (value == "Mastercard") {
            color <- "#e06631"
          } else if (value == "VISA 16 digit") {
            color <- "#0c13cf"
          } else if (value == "American Express") {
            color <- "#4d8be8"
          } else if (value == "JCB 16 digit") {
            color <- "#23c45e"
          } else {
            color <- "#777"
          }
          list(color = color, fontWeight = "bold")
        }
      )
    ),
    bordered = TRUE,
    compact = TRUE,
    searchable = TRUE,
    highlight = TRUE
  )
)

output$result <- renderPrint({
  str(edited_r())
})
```

```
}

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

`filter-data` *Shiny module to interactively filter a* `data.frame`

Description

Module generate inputs to filter `data.frame` according column's type. Code to reproduce the filter is returned as an expression with filtered data.

Usage

```
filter_data_ui(id, show_nrow = TRUE, max_height = NULL)

filter_data_server(
  id,
  data = reactive(NULL),
  vars = reactive(NULL),
  name = reactive("data"),
  defaults = reactive(NULL),
  drop_ids = getOption("datamods.filter.drop_ids", default = TRUE),
  widget_char = c("virtualSelect", "select", "picker"),
  widget_num = c("slider", "range"),
  widget_date = c("slider", "range"),
  label_na = "NA",
  value_na = TRUE
)
```

Arguments

`id`	Module id. See `shiny::moduleServer()`.
`show_nrow`	Show number of filtered rows and total.
`max_height`	Maximum height for filters panel, useful if you have many variables to filter and limited space.
`data`	`shiny::reactive()` function returning a `data.frame` to filter.
`vars`	`shiny::reactive()` function returning a `character` vector of variables for which to add a filter. If a named `list`, names are used as labels.
`name`	`shiny::reactive()` function returning a `character` string representing `data` name, only used for code generated.
`defaults`	`shiny::reactive()` function returning a named `list` of variable:value pairs which will be used to set the filters.

drop_ids	Drop columns containing more than 90% of unique values, or than 50 distinct values. Use FALSE to disable or use list(p = 0.9, n = 50) to customize threshold values.
widget_char	Widget to use for character variables: shinyWidgets::pickerInput() or shiny::selectInput() (default).
widget_num	Widget to use for numeric variables: shinyWidgets::numericRangeInput() or shiny::sliderInput() (default).
widget_date	Widget to use for date/time variables: shiny::dateRangeInput() or shiny::sliderInput() (default).
label_na	Label for missing value widget.
value_na	Default value for all NA's filters.

Value

- UI: HTML tags that can be included in shiny's UI
- Server: a list with four slots:
  - **filtered**: a reactive function returning the data filtered.
  - **code**: a reactive function returning the dplyr pipeline to filter data.
  - **expr**: a reactive function returning an expression to filter data.
  - **values**: a reactive function returning a named list of variables and filter values.

Examples

```
library(shiny)
library(shinyWidgets)
library(datamods)
library(MASS)

# Add some NAs to mpg
mtcars_na <- mtcars
mtcars_na[] <- lapply(
  X = mtcars_na,
  FUN = function(x) {
    x[sample.int(n = length(x), size = sample(5:10, 1))] <- NA
    x
  }
)

datetime <- data.frame(
  date = seq(Sys.Date(), by = "day", length.out = 300),
  datetime = seq(Sys.time(), by = "hour", length.out = 300),
  num = sample.int(1e5, 300)
)

one_column_numeric <- data.frame(
  var1 = rnorm(100)
)

ui <- fluidPage(
```

```
  tags$h2("Filter data.frame"),
  actionButton("saveFilterButton","Save Filter Values"),
  actionButton("loadFilterButton","Load Filter Values"),
  radioButtons(
    inputId = "dataset",
    label = "Data:",
    choices = c(
      "iris",
      "mtcars",
      "mtcars_na",
      "Cars93",
      "datetime",
      "one_column_numeric"
    ),
    inline = TRUE
  ),

  fluidRow(
    column(
      width = 3,
      filter_data_ui("filtering", max_height = "500px")
    ),
    column(
      width = 9,
      progressBar(
        id = "pbar", value = 100,
        total = 100, display_pct = TRUE
      ),
      reactable::reactableOutput(outputId = "table"),
      tags$b("Code dplyr:"),
      verbatimTextOutput(outputId = "code_dplyr"),
      tags$b("Expression:"),
      verbatimTextOutput(outputId = "code"),
      tags$b("Filtered data:"),
      verbatimTextOutput(outputId = "res_str")
    )
  )
)

server <- function(input, output, session) {
  savedFilterValues <- reactiveVal()
  data <- reactive({
    get(input$dataset)
  })

  vars <- reactive({
    if (identical(input$dataset, "mtcars")) {
      setNames(as.list(names(mtcars)[1:5]), c(
        "Miles/(US) gallon",
        "Number of cylinders",
        "Displacement (cu.in.)",
        "Gross horsepower",
        "Rear axle ratio"
```

```
      ))
    } else {
      NULL
    }
  })

  observeEvent(input$saveFilterButton,{
    savedFilterValues <<- res_filter$values()
  },ignoreInit = T)

  defaults <- reactive({
    input$loadFilterButton
    savedFilterValues
  })

  res_filter <- filter_data_server(
    id = "filtering",
    data = data,
    name = reactive(input$dataset),
    vars = vars,
    defaults = defaults,
    widget_num = "slider",
    widget_date = "slider",
    label_na = "Missing"
  )

  observeEvent(res_filter$filtered(), {
    updateProgressBar(
      session = session, id = "pbar",
      value = nrow(res_filter$filtered()), total = nrow(data())
    )
  })

  output$table <- reactable::renderReactable({
    reactable::reactable(res_filter$filtered())
  })


  output$code_dplyr <- renderPrint({
    res_filter$code()
  })
  output$code <- renderPrint({
    res_filter$expr()
  })

  output$res_str <- renderPrint({
    str(res_filter$filtered())
  })

}

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

get_data_packages *Get packages containing datasets*

Description

Get packages containing datasets

Usage

```
get_data_packages()
```

Value

a character vector of packages names

Examples

```
if (interactive()) {

  get_data_packages()

}
```

i18n *Internationalization*

Description

Simple mechanism to translate labels in a Shiny application.

Usage

```
i18n(x, translations = i18n_translations())

i18n_translations(package = packageName(parent.frame(2)))

set_i18n(value, packages = c("datamods", "esquisse"))
```

Arguments

x	Label to translate.
translations	Either a list or a data.frame with translations.
package	Name of the package where the function is called, use NULL outside a package. It will retrieve option "i18n.<PACKAGE>" (or "i18n" if no package) to returns appropriate labels.

value	Value to set for translation. Can be: • single character to use a supported language ("fr", "mk", "al", "pt" for esquisse and datamods packages). • a list with labels as names and translations as values. • a data.frame with 2 column: label & translation. • path to a CSV file with same structure as for data.frame above.
packages	Name of packages for which to set i18n, default to esquisse and datamods

Value

i18n() returns a character, i18n_translations() returns a list or a data.frame.

Examples

```
library(datamods)

# Use with an objet
my.translations <- list(
  "Hello" = "Bonjour"
)
i18n("Hello", my.translations)

# Use with options()
options("i18n" = list(
  "Hello" = "Bonjour"
))
i18n("Hello")

# With a package
options("datamods.i18n" = "fr")
i18n("Browse...", translations = i18n_translations("datamods"))
# If you call i18n() from within a function of your package
# you don't need second argument, e.g.:
# i18n("Browse...")
```

---

import-copypaste *Import data with copy & paste*

Description

Let the user copy data from Excel or text file then paste it into a text area to import it.

Usage

```
import_copypaste_ui(id, title = TRUE, name_field = TRUE)

import_copypaste_server(
  id,
```

```
  btn_show_data = TRUE,
  show_data_in = c("popup", "modal"),
  trigger_return = c("button", "change"),
  return_class = c("data.frame", "data.table", "tbl_df", "raw"),
  reset = reactive(NULL),
  fread_args = list()
)
```

Arguments

id	Module's ID.
title	Module's title, if TRUE use the default title, use NULL for no title or a shiny.tag for a custom one.
name_field	Show or not a field to add a name to data (that is returned server-side).
btn_show_data	Display or not a button to display data in a modal window if import is successful.
show_data_in	Where to display data: in a "popup" or in a "modal" window.
trigger_return	When to update selected data: "button" (when user click on button) or "change" (each time user select a dataset in the list).
return_class	Class of returned data: data.frame, data.table, tbl_df (tibble) or raw.
reset	A reactive function that when triggered resets the data.
fread_args	list of additional arguments to pass to data.table::fread() when reading data.

Value

- UI: HTML tags that can be included in shiny's UI
- Server: a list with three slots:
  - **status**: a reactive function returning the status: NULL, error or success.
  - **name**: a reactive function returning the name of the imported data as character.
  - **data**: a reactive function returning the imported data.frame.

Examples

```
library(shiny)
library(datamods)

ui <- fluidPage(
  tags$h3("Import data with copy & paste"),
  fluidRow(
    column(
      width = 4,
      import_copypaste_ui("myid")
    ),
    column(
      width = 8,
      tags$b("Import status:"),
      verbatimTextOutput(outputId = "status"),
```

```
      tags$b("Name:"),
      verbatimTextOutput(outputId = "name"),
      tags$b("Data:"),
      verbatimTextOutput(outputId = "data")
    )
  )
)

server <- function(input, output, session) {

  imported <- import_copypaste_server("myid")

  output$status <- renderPrint({
    imported$status()
  })
  output$name <- renderPrint({
    imported$name()
  })
  output$data <- renderPrint({
    imported$data()
  })

}

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

`import-file` *Import data from a file*

Description

Let user upload a file and import data

Usage

```
import_file_ui(
  id,
  title = TRUE,
  preview_data = TRUE,
  file_extensions = c(".csv", ".txt", ".xls", ".xlsx", ".rds", ".fst", ".sas7bdat",
    ".sav"),
  layout_params = c("dropdown", "inline")
)

import_file_server(
  id,
  btn_show_data = TRUE,
  show_data_in = c("popup", "modal"),
```

```
  trigger_return = c("button", "change"),
  return_class = c("data.frame", "data.table", "tbl_df", "raw"),
  reset = reactive(NULL),
  read_fns = list()
)
```

Arguments

id	Module's ID.
title	Module's title, if TRUE use the default title, use NULL for no title or a shiny.tag for a custom one.
preview_data	Show or not a preview of the data under the file input.
file_extensions	File extensions accepted by shiny::fileInput(), can also be MIME type.
layout_params	How to display import parameters : in a dropdown button or inline below file input.
btn_show_data	Display or not a button to display data in a modal window if import is successful.
show_data_in	Where to display data: in a "popup" or in a "modal" window.
trigger_return	When to update selected data: "button" (when user click on button) or "change" (each time user select a dataset in the list).
return_class	Class of returned data: data.frame, data.table, tbl_df (tibble) or raw.
reset	A reactive function that when triggered resets the data.
read_fns	Named list with custom function(s) to read data: • the name must be the extension of the files to which the function will be applied • the value must be a function that can have 5 arguments (you can ignore some of them, but you have to use the same names), passed by user through the interface: – file: path to the file – sheet: for Excel files, sheet to read – skip: number of row to skip – dec: decimal separator – encoding: file encoding – na.strings: character(s) to interpret as missing values.

Value

- UI: HTML tags that can be included in shiny's UI
- Server: a list with three slots:
  - **status**: a reactive function returning the status: NULL, error or success.
  - **name**: a reactive function returning the name of the imported data as character.
  - **data**: a reactive function returning the imported data.frame.

Examples

```
library(shiny)
library(datamods)

ui <- fluidPage(
  # theme = bslib::bs_theme(version = 5L),
  # theme = bslib::bs_theme(version = 5L, preset = "bootstrap"),
  tags$h3("Import data from a file"),
  fluidRow(
    column(
      width = 4,
      import_file_ui(
        id = "myid",
        file_extensions = c(".csv", ".txt", ".xls", ".xlsx", ".json"),
        layout_params = "inline" # or "dropdown"
      )
    ),
    column(
      width = 8,
      tags$b("Import status:"),
      verbatimTextOutput(outputId = "status"),
      tags$b("Name:"),
      verbatimTextOutput(outputId = "name"),
      tags$b("Code:"),
      verbatimTextOutput(outputId = "code"),
      tags$b("Data:"),
      verbatimTextOutput(outputId = "data")
    )
  )
)

server <- function(input, output, session) {

  imported <- import_file_server(
    id = "myid",
    # Custom functions to read data
    read_fns = list(
      xls = function(file, sheet, skip, encoding) {
        readxl::read_xls(path = file, sheet = sheet, skip = skip)
      },
      json = function(file) {
        jsonlite::read_json(file, simplifyVector = TRUE)
      }
    ),
    show_data_in = "modal"
  )

  output$status <- renderPrint({
    imported$status()
  })
  output$name <- renderPrint({
```

```
    imported$name()
  })
  output$code <- renderPrint({
    imported$code()
  })
  output$data <- renderPrint({
    imported$data()
  })

}

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

`import-globalenv` *Import data from an Environment*

Description

Let the user select a dataset from its own environment or from a package's environment.

Usage

```
import_globalenv_ui(
  id,
  globalenv = TRUE,
  packages = get_data_packages(),
  title = TRUE
)

import_globalenv_server(
  id,
  btn_show_data = TRUE,
  show_data_in = c("popup", "modal"),
  trigger_return = c("button", "change"),
  return_class = c("data.frame", "data.table", "tbl_df", "raw"),
  reset = reactive(NULL)
)
```

Arguments

`id`	Module's ID.
`globalenv`	Search for data in Global environment.
`packages`	Name of packages in which to search data.
`title`	Module's title, if `TRUE` use the default title, use `NULL` for no title or a `shiny.tag` for a custom one.
`btn_show_data`	Display or not a button to display data in a modal window if import is successful.

show_data_in	Where to display data: in a "popup" or in a "modal" window.
trigger_return	When to update selected data: "button" (when user click on button) or "change" (each time user select a dataset in the list).
return_class	Class of returned data: data.frame, data.table, tbl_df (tibble) or raw.
reset	A reactive function that when triggered resets the data.

Value

- UI: HTML tags that can be included in shiny's UI
- Server: a list with three slots:
  - **status**: a reactive function returning the status: NULL, error or success.
  - **name**: a reactive function returning the name of the imported data as character.
  - **data**: a reactive function returning the imported data.frame.

Examples

```
if (interactive()) {
  library(shiny)
  library(datamods)

  # Create some data.frames

  my_df <- data.frame(
    variable1 = sample(letters, 20, TRUE),
    variable2 = sample(1:100, 20, TRUE)
  )

  results_analysis <- data.frame(
    id = sample(letters, 20, TRUE),
    measure = sample(1:100, 20, TRUE),
    response = sample(1:100, 20, TRUE)
  )


  # Application

  ui <- fluidPage(
    fluidRow(
      column(
        width = 4,
        import_globalenv_ui("myid")
      ),
      column(
        width = 8,
        tags$b("Import status:"),
        verbatimTextOutput(outputId = "status"),
        tags$b("Name:"),
        verbatimTextOutput(outputId = "name"),
        tags$b("Data:"),
        verbatimTextOutput(outputId = "data")
```

```
    )
  )
)

server <- function(input, output, session) {

  imported <- import_globalenv_server("myid")

  output$status <- renderPrint({
    imported$status()
  })
  output$name <- renderPrint({
    imported$name()
  })
  output$data <- renderPrint({
    imported$data()
  })

}

shinyApp(ui, server)
}
```

`import-googlesheets` *Import data from Googlesheet*

Description

Let user paste link to a Google sheet then import the data.

Usage

```
import_googlesheets_ui(id, title = TRUE)

import_googlesheets_server(
  id,
  btn_show_data = TRUE,
  show_data_in = c("popup", "modal"),
  trigger_return = c("button", "change"),
  return_class = c("data.frame", "data.table", "tbl_df", "raw"),
  reset = reactive(NULL)
)
```

Arguments

`id`	Module's ID.
`title`	Module's title, if `TRUE` use the default title, use `NULL` for no title or a `shiny.tag` for a custom one.

btn_show_data	Display or not a button to display data in a modal window if import is successful.
show_data_in	Where to display data: in a "popup" or in a "modal" window.
trigger_return	When to update selected data: "button" (when user click on button) or "change" (each time user select a dataset in the list).
return_class	Class of returned data: data.frame, data.table, tbl_df (tibble) or raw.
reset	A reactive function that when triggered resets the data.

Value

- UI: HTML tags that can be included in shiny's UI
- Server: a list with three slots:
  - **status**: a reactive function returning the status: NULL, error or success.
  - **name**: a reactive function returning the name of the imported data as character.
  - **data**: a reactive function returning the imported data.frame.

Examples

```
library(shiny)
library(datamods)

ui <- fluidPage(
  tags$h3("Import data from Googlesheets"),
  fluidRow(
    column(
      width = 4,
      import_googlesheets_ui("myid")
    ),
    column(
      width = 8,
      tags$b("Import status:"),
      verbatimTextOutput(outputId = "status"),
      tags$b("Name:"),
      verbatimTextOutput(outputId = "name"),
      tags$b("Data:"),
      verbatimTextOutput(outputId = "data")
    )
  )
)

server <- function(input, output, session) {

  imported <- import_googlesheets_server("myid")

  output$status <- renderPrint({
    imported$status()
  })
  output$name <- renderPrint({
    imported$name()
  })
  output$data <- renderPrint({
```

```
    imported$data()
  })

}

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

`import-modal` *Import from all sources*

Description

Wrap all import modules into one, can be displayed inline or in a modal window..

Usage

```
import_ui(
  id,
  from = c("env", "file", "copypaste", "googlesheets", "url"),
 file_extensions = c(".csv", ".txt", ".xls", ".xlsx", ".rds", ".fst", ".sas7bdat",
    ".sav")
)

import_server(
  id,
  validation_opts = NULL,
  allowed_status = c("OK", "Failed", "Error"),
  return_class = c("data.frame", "data.table", "tbl_df", "raw"),
  read_fns = list()
)

import_modal(
  id,
  from,
  title = i18n("Import data"),
  size = "l",
 file_extensions = c(".csv", ".txt", ".xls", ".xlsx", ".rds", ".fst", ".sas7bdat",
    ".sav")
)
```

Arguments

`id`	Module's id
`from`	The import_ui & server to use, i.e. the method. There are 5 options to choose from. ("env", "file", "copypaste", "googlesheets", "url")
`file_extensions`	File extensions accepted by `shiny::fileInput()`, can also be MIME type.

validation_opts
: list of arguments passed to [validation_server().

allowed_status
: Vector of statuses allowed to confirm dataset imported, if you want that all validation rules are successful before importing data use allowed_status = "OK".

return_class
: Class of returned data: data.frame, data.table, tbl_df (tibble) or raw.

read_fns
: Named list with custom function(s) to read data:

  - the name must be the extension of the files to which the function will be applied
  - the value must be a function that can have 5 arguments (you can ignore some of them, but you have to use the same names), passed by user through the interface:
    - file: path to the file
    - sheet: for Excel files, sheet to read
    - skip: number of row to skip
    - dec: decimal separator
    - encoding: file encoding
    - na.strings: character(s) to interpret as missing values.

title
: Modal window title.

size
: Modal window size, default to "l" (large).

Value

- UI: HTML tags that can be included in shiny's UI
- Server: a list with three slots:
  - **status**: a reactive function returning the status: NULL, error or success.
  - **name**: a reactive function returning the name of the imported data as character.
  - **data**: a reactive function returning the imported data.frame.

Examples

```
library(shiny)
library(datamods)

ui <- fluidPage(
  # Try with different Bootstrap version
  theme = bslib::bs_theme(version = 5, preset = "bootstrap"),
  fluidRow(
    column(
      width = 4,
      checkboxGroupInput(
        inputId = "from",
        label = "From",
        choices = c("env", "file", "copypaste", "googlesheets", "url"),
        selected = c("file", "copypaste")
      ),
      actionButton("launch_modal", "Launch modal window")
    ),
```

```
    column(
      width = 8,
      tags$b("Imported data:"),
      verbatimTextOutput(outputId = "name"),
      verbatimTextOutput(outputId = "data"),
      verbatimTextOutput(outputId = "str_data")
    )
  )
)

server <- function(input, output, session) {

  observeEvent(input$launch_modal, {
    req(input$from)
    import_modal(
      id = "myid",
      from = input$from,
      title = "Import data to be used in application"
    )
  })

  imported <- import_server("myid", return_class = "tbl_df")

  output$name <- renderPrint({
    req(imported$name())
    imported$name()
  })

  output$data <- renderPrint({
    req(imported$data())
    imported$data()
  })

  output$str_data <- renderPrint({
    req(imported$data())
    str(imported$data())
  })

}

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

`import-url` *Import data from a URL*

Description

Let user paste link to a JSON then import the data.

Usage

```
import_url_ui(id, title = TRUE)

import_url_server(
  id,
  btn_show_data = TRUE,
  show_data_in = c("popup", "modal"),
  trigger_return = c("button", "change"),
  return_class = c("data.frame", "data.table", "tbl_df", "raw"),
  reset = reactive(NULL)
)
```

Arguments

id	Module's ID.
title	Module's title, if TRUE use the default title, use NULL for no title or a shiny.tag for a custom one.
btn_show_data	Display or not a button to display data in a modal window if import is successful.
show_data_in	Where to display data: in a "popup" or in a "modal" window.
trigger_return	When to update selected data: "button" (when user click on button) or "change" (each time user select a dataset in the list).
return_class	Class of returned data: data.frame, data.table, tbl_df (tibble) or raw.
reset	A reactive function that when triggered resets the data.

Value

- UI: HTML tags that can be included in shiny's UI
- Server: a list with three slots:
  - **status**: a reactive function returning the status: NULL, error or success.
  - **name**: a reactive function returning the name of the imported data as character.
  - **data**: a reactive function returning the imported data.frame.

Examples

```
library(shiny)
library(datamods)

ui <- fluidPage(
  tags$h3("Import data from URL"),
  fluidRow(
    column(
      width = 4,
      import_url_ui("myid")
    ),
    column(
      width = 8,
      tags$b("Import status:"),
```

```
      verbatimTextOutput(outputId = "status"),
      tags$b("Name:"),
      verbatimTextOutput(outputId = "name"),
      tags$b("Data:"),
      verbatimTextOutput(outputId = "data")
    )
  )
)

server <- function(input, output, session) {

  imported <- import_url_server(
    "myid",
    btn_show_data = FALSE,
    return_class = "raw"
  )

  output$status <- renderPrint({
    imported$status()
  })
  output$name <- renderPrint({
    imported$name()
  })
  output$data <- renderPrint({
    imported$data()
  })

}

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

`list_pkg_data` *List dataset contained in a package*

Description

List dataset contained in a package

Usage

```
list_pkg_data(pkg)
```

Arguments

`pkg`	Name of the package, must be installed.

Value

a `character` vector or `NULL`.

Examples

```
list_pkg_data("ggplot2")
```

`module-sample` *Shiny module to interactively sample a* `data.frame`

Description

Allow to take a sample of `data.frame` for a given number or proportion of rows to keep.

Usage

```
sample_ui(id)

sample_server(id, data_r = reactive(NULL))
```

Arguments

`id`	Module id. See `shiny::moduleServer()`.
`data_r`	`reactive` containing a `data.frame` to use in the module.

Value

- UI: HTML tags that can be included in shiny's UI
- Server: a `reactive` fgunction with the sampled data.

Examples

```
library(shiny)
library(datamods)
library(reactable)


ui <- fluidPage(

  tags$h2("Sampling"),

  fluidRow(
    column(
      width = 3,
      sample_ui("myID")
    ),
    column(
      width = 9,
      reactableOutput("table")
    )
  )
)
```

```
server <- function(input, output, session) {

  result_sample <- sample_server("myID", reactive(iris))

  output$table <- renderReactable({
    table_sample <- reactable(
      data = result_sample(),
      defaultColDef = colDef(
        align = "center"
      ),
      borderless = TRUE,
      highlight = TRUE,
      striped = TRUE
    )
    return(table_sample)
  })
}

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

`select-group` *Select Group Input Module*

Description

Group of mutually dependent select menus for filtering `data.frame`'s columns (like in Excel).

Usage

```
select_group_ui(
  id,
  params,
  label = NULL,
  btn_reset_label = "Reset filters",
  inline = TRUE,
  vs_args = list()
)

select_group_server(id, data_r, vars_r)
```

Arguments

`id`	Module's id.
`params`	A list of parameters passed to each `shinyWidgets::virtualSelectInput()`, you can use :

- `inputId`: mandatory, must correspond to variable name.
- `label`: Display label for the control.
- `placeholder`: Text to show when no options selected.

`label` Character, global label on top of all labels.

`btn_reset_label`
Character, reset button label. If `NULL` no button is added.

`inline` If `TRUE` (the default), select menus are horizontally positioned, otherwise vertically.

`vs_args` Arguments passed to all `shinyWidgets::virtualSelectInput()` created.

`data_r` Either a `data.frame()` or a `shiny::reactive()` function returning a `data.frame` (do not use parentheses).

`vars_r` character, columns to use to create filters, must correspond to variables listed in `params`. Can be a `shiny::reactive()` function, but values must be included in the initial ones (in `params`).

Value

A `shiny::reactive()` function containing data filtered with an attribute `inputs` containing a named list of selected inputs.

Examples

```
# Default ------------------------------------------------------------------

library(shiny)
library(datamods)
library(shinyWidgets)


ui <- fluidPage(
  # theme = bslib::bs_theme(version = 5L),
  fluidRow(
    column(
      width = 10, offset = 1,
      tags$h3("Filter data with select group module"),
      shinyWidgets::panel(
        select_group_ui(
          id = "my-filters",
          params = list(
            list(inputId = "Manufacturer", label = "Manufacturer:"),
            list(inputId = "Type", label = "Type:"),
            list(inputId = "AirBags", label = "AirBags:"),
            list(inputId = "DriveTrain", label = "DriveTrain:")
          ), vs_args = list(disableSelectAll = FALSE)
        ),
        status = "primary"
      ),
      reactable::reactableOutput(outputId = "table"),
      tags$b("Inputs values:"),
```

```
      verbatimTextOutput("inputs")
    )
  )
)

server <- function(input, output, session) {
  res_mod <- select_group_server(
    id = "my-filters",
    data = reactive(MASS::Cars93),
    vars = reactive(c("Manufacturer", "Type", "AirBags", "DriveTrain"))
  )

  output$table <- reactable::renderReactable({
    reactable::reactable(res_mod())
  })

  output$inputs <- renderPrint({
    attr(res_mod(), "inputs")
  })
}

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

---

show_data *Display a table in a window*

Description

Display a table in a window

Usage

```
show_data(
  data,
  title = NULL,
  options = NULL,
  show_classes = TRUE,
  type = c("popup", "modal", "winbox"),
  width = "65%",
  ...
)
```

Arguments

data	a data object (either a `matrix` or a `data.frame`).
title	Title to be displayed in window.
options	Arguments passed to `toastui::datagrid()`.

show_classes	Show variables classes under variables names in table header.
type	Display table in a pop-up with shinyWidgets::show_alert(), in modal window with shiny::showModal() or in a WinBox window with shinyWidgets::WinBox().
width	Width of the window, only used if type = "popup" or type = "winbox".
...	Additional options, such as wbOptions = wbOptions() or wbControls = wbControls().

Value

No value.

Note

If you use type = "winbox", you'll need to use shinyWidgets::html_dependency_winbox() somewhere in your UI.

Examples

```
library(shiny)
library(datamods)

ui <- fluidPage(
  theme = bslib::bs_theme(version = 5L),
  shinyWidgets::html_dependency_winbox(),
  actionButton(
    inputId = "show1",
    label = "Show data in popup",
    icon = icon("eye")
  ),
  actionButton(
    inputId = "show2",
    label = "Show data in modal",
    icon = icon("eye")
  ),
  actionButton(
    inputId = "show3",
    label = "Show data without classes",
    icon = icon("eye")
  ),
  actionButton(
    inputId = "show4",
    label = "Show data in Winbox",
    icon = icon("eye")
  )
)

server <- function(input, output, session) {
  observeEvent(input$show1, {
    show_data(MASS::Cars93, title = "MASS::Cars93 dataset", type = "popup")
  })
  observeEvent(input$show2, {
    show_data(MASS::Cars93, title = "MASS::Cars93 dataset", type = "modal")
```

```
  })
  observeEvent(input$show3, {
    show_data(
      data = MASS::Cars93,
      title = "MASS::Cars93 dataset",
      show_classes = FALSE,
      options = list(pagination = 10),
      type = "modal"
    )
  })
  observeEvent(input$show4, {
    show_data(
      MASS::Cars93,
      title = "MASS::Cars93 dataset",
      type = "winbox",
      wbOptions = shinyWidgets::wbOptions(background = "forestgreen")
    )
  })
}

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

`update-factor` *Module to Reorder the Levels of a Factor Variable*

Description

This module contain an interface to reorder the levels of a factor variable.

Usage

```
update_factor_ui(id)

update_factor_server(id, data_r = reactive(NULL))

modal_update_factor(
  id,
  title = i18n("Update levels of a factor"),
  easyClose = TRUE,
  size = "l",
  footer = NULL
)
```

Arguments

`id`	Module ID.
`data_r`	A `shiny::reactive()` function returning a `data.frame`.

title	An optional title for the dialog.
easyClose	If TRUE, the modal dialog can be dismissed by clicking outside the dialog box, or be pressing the Escape key. If FALSE (the default), the modal dialog can't be dismissed in those ways; instead it must be dismissed by clicking on a modalButton(), or from a call to removeModal() on the server.
size	One of "s" for small, "m" (the default) for medium, "l" for large, or "xl" for extra large. Note that "xl" only works with Bootstrap 4 and above (to opt-in to Bootstrap 4+, pass bslib::bs_theme() to the theme argument of a page container like fluidPage()).
footer	UI for footer. Use NULL for no footer.

Value

A shiny::reactive() function returning the data.

Examples

```
library(shiny)
library(datamods)
library(ggplot2)

ui <- fluidPage(
  theme = bslib::bs_theme(version = 5L, preset = "bootstrap"),
  shinyWidgets::html_dependency_winbox(),
  tags$h2("Reorder the Levels of a Factor"),
  fluidRow(
    column(
      width = 6,
      update_factor_ui("id"),
      actionButton("modal", "Or click here to open a modal to update factor's level"),
      tags$br(), tags$br(),
      actionButton("winbox", "Or click here to open a WinBox to create a column")
    ),
    column(
      width = 6,
      selectInput(
        "var",
        label = "Variable to plot:",
        choices = NULL
      ),
      plotOutput("plot"),
      verbatimTextOutput("res")
    )
  )
)

server <- function(input, output, session) {

  rv <- reactiveValues(data = MASS::Cars93[c(1, 2, 3, 9, 10, 11, 16, 26, 27)])
  observe(
    updateSelectInput(inputId = "var", choices = names(rv$data))
```

```
  )

  # Inline mode
  data_inline_r <- update_factor_server(
    id = "id",
    data_r = reactive(rv$data)
  )
  observeEvent(data_inline_r(), rv$data <- data_inline_r())

  # modal window mode
  observeEvent(input$modal, modal_update_factor("modal"))
  data_modal_r <- update_factor_server(
    id = "modal",
    data_r = reactive(rv$data)
  )
  observeEvent(data_modal_r(), {
    shiny::removeModal()
    rv$data <- data_modal_r()
  })

  # winbox mode
  observeEvent(input$winbox, winbox_update_factor("winbox"))
  data_winbox_r <- update_factor_server(
    id = "winbox",
    data_r = reactive(rv$data)
  )
  observeEvent(data_winbox_r(), rv$data <- data_winbox_r())

  # Plot results
  output$plot <- renderPlot({
    req(input$var, rv$data)
    ggplot(rv$data) +
      aes(x = !!sym(input$var)) +
      geom_bar()
  })
  # Show results
  output$res <- renderPrint({
    data <- req(rv$data)
    str(data)
  })
}

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

---

update-variables *Select, rename and convert variables*

Description

Select, rename and convert variables

Usage

```
update_variables_ui(id, title = TRUE)

update_variables_server(
  id,
  data,
  height = NULL,
  return_data_on_init = FALSE,
  try_silent = FALSE
)
```

Arguments

id	Module's ID
title	Module's title, if TRUE use the default title, use NULL for no title or a shiny.tag for a custom one.
data	a data.frame or a reactive function returning a data.frame.
height	Height for the table.
return_data_on_init	Return initial data when module is called.
try_silent	logical: should the report of error messages be suppressed?

Value

A shiny::reactive() function returning the updated data.

Examples

```
library(shiny)
library(datamods)

testdata <- data.frame(
  date_as_char = as.character(Sys.Date() + 0:9),
  date_as_num = as.numeric(Sys.Date() + 0:9),
  datetime_as_char = as.character(Sys.time() + 0:9 * 3600*24),
  datetime_as_num = as.numeric(Sys.time() + 0:9 * 3600*24),
  num_as_char = as.character(1:10),
  char = month.name[1:10],
  char_na = c("A", "A", "B", NA, "B", "A", NA, "B", "A", "B"),
  stringsAsFactors = FALSE
)

ui <- fluidPage(
  theme = bslib::bs_theme(version = 5L, preset = "bootstrap"),
  tags$h3("Select, rename and convert variables"),
  fluidRow(
    column(
      width = 6,
      # radioButtons()
```

```
      update_variables_ui("vars")
    ),
    column(
      width = 6,
      tags$b("original data:"),
      verbatimTextOutput("original"),
      verbatimTextOutput("original_str"),
      tags$b("Modified data:"),
      verbatimTextOutput("modified"),
      verbatimTextOutput("modified_str")
    )
  )
)

server <- function(input, output, session) {

  updated_data <- update_variables_server(
    id = "vars",
    data = reactive(testdata),
    return_data_on_init = FALSE
  )

  output$original <- renderPrint({
    testdata
  })
  output$original_str <- renderPrint({
    str(testdata)
  })

  output$modified <- renderPrint({
    updated_data()
  })
  output$modified_str <- renderPrint({
    str(updated_data())
  })
}

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

`validation_ui` *Validation module*

Description

Check that a dataset respect some validation expectations.

Usage

```
validation_ui(id, display = c("dropdown", "inline"), max_height = NULL, ...)
```

```
validation_server(
  id,
  data,
  n_row = NULL,
  n_col = NULL,
  n_row_label = i18n("Valid number of rows"),
  n_col_label = i18n("Valid number of columns"),
  btn_label = i18n("Dataset validation:"),
  rules = NULL,
  bs_version = 3
)
```

Arguments

id	Module's ID.
display	Display validation results in a dropdown menu by clicking on a button or display results directly in interface.
max_height	Maximum height for validation results element, useful if you have many rules.
...	Arguments passed to actionButton or uiOutput depending on display mode, you cannot use inputId/outputId, label or icon (button only).
data	a reactive function returning a data.frame.
n_row, n_col	A one-sided formula to check number of rows and columns respectively, see below for examples.
n_row_label, n_col_label	Text to be displayed with the result of the check for number of rows/columns.
btn_label	Label for the dropdown button, will be followed by validation result.
rules	An object of class validator created with validate::validator.
bs_version	Bootstrap version used, it may affect rendering, especially status badges.

Value

- UI: HTML tags that can be included in shiny's UI
- Server: a list with two slots:
  - **status**: a reactive function returning the best status available between "OK", "Failed" or "Error".
  - **details**: a reactive function returning a list with validation details.

Examples

```
library(datamods)
library(shiny)

if (requireNamespace("validate")) {
  library(validate)
```

```
# Define some rules to be applied to data
myrules <- validator(
  is.character(Manufacturer) | is.factor(Manufacturer),
  is.numeric(Price),
  Price > 12, # we should use 0 for testing positivity, but that's for the example
  !is.na(Luggage.room),
  in_range(Cylinders, min = 4, max = 8),
  Man.trans.avail %in% c("Yes", "No")
)
# Add some labels
label(myrules) <- c(
  "Variable Manufacturer must be character",
  "Variable Price must be numeric",
  "Variable Price must be strictly positive",
  "Luggage.room must not contain any missing values",
  "Cylinders must be between 4 and 8",
  "Man.trans.avail must be 'Yes' or 'No'"
)
# you can also add a description()

ui <- fluidPage(
  tags$h2("Validation"),
  fluidRow(
    column(
      width = 4,
      radioButtons(
        inputId = "dataset",
        label = "Choose dataset:",
        choices = c("mtcars", "MASS::Cars93")
      ),
      tags$p("Dropdown example:"),
      validation_ui("validation1"),

      tags$br(),

      tags$p("Inline example:"),
      validation_ui("validation2", display = "inline")
    ),
    column(
      width = 8,
      tags$b("Status:"),
      verbatimTextOutput("status"),
      tags$b("Details:"),
      verbatimTextOutput("details")
    )
  )
)

server <- function(input, output, session) {

  dataset <- reactive({
    if (input$dataset == "mtcars") {
      mtcars
```

```
    } else {
      MASS::Cars93
    }
  })

  results <- validation_server(
    id = "validation1",
    data = dataset,
    n_row = ~ . > 20, # more than 20 rows
    n_col = ~ . >= 3, # at least 3 columns
    rules = myrules
  )

  validation_server(
    id = "validation2",
    data = dataset,
    n_row = ~ . > 20, # more than 20 rows
    n_col = ~ . >= 3, # at least 3 columns
    rules = myrules
  )

  output$status <- renderPrint(results$status())
  output$details <- renderPrint(results$details())

}

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

Index