

Package ‘datamods’

September 26, 2023

Title Modules to Import and Manipulate Data in 'Shiny'

Version 1.4.2

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>

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

Encoding UTF-8

RoxygenNote 7.2.3

Imports data.table, htmltools, htmlwidgets, phosphoricons, reactable, readxl, rio, rlang, shiny ($\geq 1.5.0$), shinyWidgets ($\geq 0.7.3$), tibble, tools, shinybusy, writexl

Suggests bslib, 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 2023-09-26 12:50:06 UTC

R topics documented:

demo_edit	2
edit-data	3
filter-data	4
get_data_packages	8
i18n	9
import-copypaste	10
import-file	12
import-globalenv	14
import-goolesheets	16
import-modal	18
import-url	21
list_pkg_data	22
module-sample	23
select-group	24
show_data	26
update-variables	28
validation_ui	30
Index	33

demo_edit	<i>Customer Credit Card Information</i>
-----------	---

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	<i>Shiny module to interactively edit a data.frame</i>
-----------	--

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,
  return_class = c("data.frame", "data.table", "tbl_df", "raw")
)
```

Arguments

<code>id</code>	Module ID
<code>data_r</code>	<code>data_r</code> reactive function containing a <code>data.frame</code> to use in the module.
<code>add</code>	boolean, if TRUE, allows you to add a row in the table via a button at the top right
<code>update</code>	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
<code>delete</code>	boolean, if TRUE, allows a row to be deleted from the table via a button in the table
<code>download_csv</code>	if TRUE, allows to export the table in csv format via a download button
<code>download_excel</code>	if TRUE, allows to export the table in excel format via a download button
<code>file_name_export</code>	character that allows you to choose the export name of the downloaded file
<code>var_edit</code>	vector of character which allows to choose the names of the editable columns
<code>var_mandatory</code>	vector of character which allows to choose obligatory fields to fill
<code>return_class</code>	Class of returned data: <code>data.frame</code> , <code>data.table</code> , <code>tbl_df</code> (tibble) or <code>raw</code> .

Value

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

Examples

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

ui <- fluidPage(
  theme = bs_theme(
    version = 5
  ),
  tags$h2(i18n("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")
  )

  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 = TRUE,
  widget_char = c("virtualSelect", "select", "picker"),
  widget_num = c("slider", "range"),
  widget_date = c("slider", "range"),
  label_na = "NA",
  value_na = TRUE
)

```

Arguments

<code>id</code>	Module id. See <code>shiny::moduleServer()</code> .
<code>show_nrow</code>	Show number of filtered rows and total.
<code>max_height</code>	Maximum height for filters panel, useful if you have many variables to filter and limited space.
<code>data</code>	<code>shiny::reactive()</code> function returning a <code>data.frame</code> to filter.
<code>vars</code>	<code>shiny::reactive()</code> function returning a character vector of variables for which to add a filter. If a named list, names are used as labels.
<code>name</code>	<code>shiny::reactive()</code> function returning a character string representing data name, only used for code generated.
<code>defaults</code>	<code>shiny::reactive()</code> function returning a named list of variable:value pairs which will be used to set the filters.
<code>drop_ids</code>	Drop columns containing more than 90% of unique values, or than 50 distinct values.
<code>widget_char</code>	Widget to use for character variables: <code>shinyWidgets::pickerInput()</code> or <code>shiny::selectInput()</code> (default).
<code>widget_num</code>	Widget to use for numeric variables: <code>shinyWidgets::numericRangeInput()</code> or <code>shiny::sliderInput()</code> (default).
<code>widget_date</code>	Widget to use for date/time variables: <code>shiny::dateRangeInput()</code> or <code>shiny::sliderInput()</code> (default).
<code>label_na</code>	Label for missing value widget.
<code>value_na</code>	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
  }
)

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: <ul style="list-style-type: none">• 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

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

<code>name_field</code>	Show or not a field to add a name to data (that is returned server-side).
<code>btn_show_data</code>	Display or not a button to display data in a modal window if import is successful.
<code>show_data_in</code>	Where to display data: in a "popup" or in a "modal" window.
<code>trigger_return</code>	When to update selected data: "button" (when user click on button) or "change" (each time user select a dataset in the list).
<code>return_class</code>	Class of returned data: <code>data.frame</code> , <code>data.table</code> , <code>tbl_df</code> (tibble) or <code>raw</code> .
<code>reset</code>	A reactive function that when triggered resets the data.
<code>fread_args</code>	list of additional arguments to pass to <code>data.table::fread()</code> 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")
)

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

<code>id</code>	Module's ID.
<code>title</code>	Module's title, if TRUE use the default title, use NULL for no title or a shiny.tag for a custom one.
<code>preview_data</code>	Show or not a preview of the data under the file input.
<code>file_extensions</code>	File extensions accepted by <code>shiny::fileInput()</code> , can also be MIME type.

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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – file: path to the file – sheet: for Excel files, sheet to read – skip: number of row to skip – dec: decimal separator – encoding: file encoding

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$h3("Import data from a file"),
  fluidRow(
    column(
      width = 4,
      import_file_ui(
        id = "myid",
        file_extensions = c(".csv", ".txt", ".xls", ".xlsx", ".json")
      )
    ),
    column(
      width = 8,
      tags$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_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$data <- renderPrint({
    imported$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

<code>id</code>	Module's ID.
<code>globalenv</code>	Search for data in Global environment.
<code>packages</code>	Name of packages in which to search data.
<code>title</code>	Module's title, if TRUE use the default title, use NULL for no title or a shiny.tag for a custom one.
<code>btn_show_data</code>	Display or not a button to display data in a modal window if import is successful.
<code>show_data_in</code>	Where to display data: in a "popup" or in a "modal" window.
<code>trigger_return</code>	When to update selected data: "button" (when user click on button) or "change" (each time user select a dataset in the list).
<code>return_class</code>	Class of returned data: data.frame, data.table, tbl_df (tibble) or raw.
<code>reset</code>	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)
}

```


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

<code>id</code>	Module's ID.
<code>title</code>	Module's title, if TRUE use the default title, use NULL for no title or a shiny.tag for a custom one.
<code>btn_show_data</code>	Display or not a button to display data in a modal window if import is successful.
<code>show_data_in</code>	Where to display data: in a "popup" or in a "modal" window.
<code>trigger_return</code>	When to update selected data: "button" (when user click on button) or "change" (each time user select a dataset in the list).
<code>return_class</code>	Class of returned data: data.frame, data.table, tbl_df (tibble) or raw.
<code>reset</code>	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_googleworksheets_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_googleworksheets_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 = "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 <code>shiny::fileInput()</code> , can also be MIME type.
validation_opts	list of arguments passed to <code>[validation_server()]</code> .
allowed_status	Vector of statuses allowed to confirm dataset imported, if you want that all validation rules are successful before importing data use <code>allowed_status = "OK"</code> .
return_class	Class of returned data: <code>data.frame</code> , <code>data.table</code> , <code>tbl_df</code> (tibble) or <code>raw</code> .
read_fns	Named list with custom function(s) to read data: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – file: path to the file – sheet: for Excel files, sheet to read – skip: number of row to skip – dec: decimal separator – encoding: file encoding
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 = 4),
  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")
    )
  )
)

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()
    })
  }

  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

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

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	<i>Shiny module to interactively sample a data.frame</i>
---------------	--

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 function with the sampled data.

Examples

```
library(shiny)
library(reactable)

##### ui.R #####

ui <- fluidPage(

  titlePanel("Sampling"),

  fluidRow(
    column(
      width = 4,
      sample_ui("myID")
    ),
    column(
      width = 8,
      reactableOutput("table")
    )
  )
)

##### server.R #####

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)
```


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

<code>id</code>	Module's id.
<code>params</code>	A list of parameters passed to each <code>shinyWidgets::virtualSelectInput()</code> , you can use : <ul style="list-style-type: none"> • <code>inputId</code>: mandatory, must correspond to variable name. • <code>label</code>: Display label for the control. • <code>placeholder</code>: Text to show when no options selected.
<code>label</code>	Character, global label on top of all labels.
<code>btn_reset_label</code>	Character, reset button label. If NULL no button is added.
<code>inline</code>	If TRUE (the default), select menus are horizontally positioned, otherwise vertically.
<code>vs_args</code>	Arguments passed to all <code>shinyWidgets::virtualSelectInput()</code> created.
<code>data_r</code>	Either a <code>data.frame()</code> or a <code>shiny::reactive()</code> function returning a <code>data.frame</code> (do not use parentheses).
<code>vars_r</code>	character, columns to use to create filters, must correspond to variables listed in <code>params</code> . Can be a <code>shiny::reactive()</code> function, but values must be included in the initial ones (in <code>params</code>).

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$h3("Filter data with select group module"),
      shinyWidgets::panel(
        select_group_ui(
          id = "my-filters",
          params = list(
            list(inputId = "Manufacturer", label = "Manufacturer:"),
            list(inputId = "Type", label = "Type:"),
            list(inputId = "AirBags", label = "AirBags:"),
            list(inputId = "DriveTrain", label = "DriveTrain:")
          )
        ),
        status = "primary"
      ),
      reactable::reactableOutput(outputId = "table"),
      tags$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)

```

Description

Display a table in a window

Usage

```
show_data(  
  data,  
  title = NULL,  
  options = NULL,  
  show_classes = TRUE,  
  type = c("popup", "modal"),  
  width = "80%"  
)
```

Arguments

data	a data object (either a matrix or a data.frame).
title	Title to be displayed in window.
options	Arguments passed to <code>reactable::reactable()</code> .
show_classes	Show variables classes under variables names in table header.
type	Display table in a pop-up or in modal window.
width	Width of the window, only used if type = "popup".

Value

No value.

Examples

```
library(shiny)  
library(datamods)  
  
ui <- fluidPage(  
  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")  
  )  
)
```

```

)

server <- function(input, output, session) {
  observeEvent(input$show1, {
    show_data(mtcars, title = "My data")
  })
  observeEvent(input$show2, {
    show_data(mtcars, title = "My data", type = "modal")
  })
  observeEvent(input$show3, {
    show_data(
      data = mtcars,
      title = "My data",
      show_classes = FALSE,
      options = list(searchable = TRUE, highlight = TRUE)
    )
  })
}

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)
```

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.

Value

A 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(
  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)
  )

  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 = "Valid number of rows",
  n_col_label = "Valid number of columns",
  btn_label = "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 <code>actionButton</code> or <code>uiOutput</code> depending on display mode, you cannot use <code>inputId/outputId</code> , <code>label</code> or <code>icon</code> (button only).
data	a reactive function returning a <code>data.frame</code> .
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 <code>validator</code> created with <code>validate::validator</code> .
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

- * **datasets**
 - demo_edit, 2
- data.frame(), 25
- data.table::fread(), 11
- demo_edit, 2
- edit-data, 3
- edit_data_server (edit-data), 3
- edit_data_ui (edit-data), 3
- filter-data, 4
- filter_data_server (filter-data), 4
- filter_data_ui (filter-data), 4
- get_data_packages, 8
- i18n, 9
- i18n_translations (i18n), 9
- import-copypaste, 10
- import-file, 12
- import-globalenv, 14
- import-goolesheets, 16
- import-modal, 18
- import-url, 21
- import_copypaste_server
 - (import-copypaste), 10
- import_copypaste_ui (import-copypaste), 10
- import_file_server (import-file), 12
- import_file_ui (import-file), 12
- import_globalenv_server
 - (import-globalenv), 14
- import_globalenv_ui (import-globalenv), 14
- import_goolesheets_server
 - (import-goolesheets), 16
- import_goolesheets_ui
 - (import-goolesheets), 16
- import_modal (import-modal), 18
- import_server (import-modal), 18
- import_ui (import-modal), 18
- import_url_server (import-url), 21
- import_url_ui (import-url), 21
- list_pkg_data, 22
- module-sample, 23
- reactable::reactable(), 27
- sample_server (module-sample), 23
- sample_ui (module-sample), 23
- select-group, 24
- select_group_server (select-group), 24
- select_group_ui (select-group), 24
- set_i18n (i18n), 9
- shiny::dateRangeInput(), 5
- shiny::fileInput(), 12, 19
- shiny::moduleServer(), 5, 23
- shiny::reactive(), 5, 25
- shiny::selectInput(), 5
- shiny::sliderInput(), 5
- shinyWidgets::numericRangeInput(), 5
- shinyWidgets::pickerInput(), 5
- shinyWidgets::virtualSelectInput(), 25
- show_data, 26
- update-variables, 28
- update_variables_server
 - (update-variables), 28
- update_variables_ui (update-variables), 28
- validation_server (validation_ui), 30
- validation_ui, 30