Package 'shiny.tailwind'

October 13, 2022

Type Package

Title 'TailwindCSS' for Shiny Apps

Version 0.2.2

Description Allows 'TailwindCSS' to be used in Shiny apps with just-in-time compiling, custom css with '@apply' directive, and custom tailwind configurations.

Encoding UTF-8

RoxygenNote 7.2.1

License MIT + file LICENSE

Depends R (>= 3.6.0), htmltools

Imports shiny

URL https://github.com/kylebutts/shiny.tailwind

BugReports https://github.com/kylebutts/shiny.tailwind/issues

Suggests fontawesome, palmerpenguins, ggplot2

NeedsCompilation no

Author Kyle Butts [aut, cre] (<https://orcid.org/0000-0002-9048-8059>), David Zimmermann-Kollenda [ctb]

Maintainer Kyle Butts <kyle.butts@colorado.edu>

Repository CRAN

Date/Publication 2022-10-13 08:50:02 UTC

R topics documented:

`compile_tailwindcss` *Starts the 'TailwindCSS' CLI*

Description

See also tailwind docs

Usage

```
compile_tailwindcss(
  infile,
  outfile,
  watch = FALSE,
  minify = FALSE,
  content = ".",
  tailwindcss = NULL,
  verbose = FALSE
)
```

Arguments

`infile`	the 'TailwindCSS' file (eg containing the `@tailwind` directives). Relative to basedir
`outfile`	the target css file, where tailwind will write the css to. Relative to basedir
`watch`	if the files should be continuously monitored (versus only compile the css once), default is False
`minify`	if the code should be minified, default is FALSE
`content`	content paths to remove unused classes, default is current dir

tailwindcss	name and path to the executable
verbose	print information

Value

the outfile invisibly

See Also

install_tailwindcss_cli

Examples

```
if (interactive()) {
  temp <- tempdir()
  owd <- setwd(temp)

  infile <- "custom.css"
  writeLines("@tailwind base;", infile)
  outfile <- "out.css"

 # file.copy(system.file("examples", "01-Old_Faithful", "app.R", package = "shiny.tailwind"),
  #           "app.R", overwrite = TRUE)

  # write a mini shiny UI
  writeLines("
    library(shiny)
    div(class = \"page-div\",
        div(class = \"w-full text-center py-12\",
            h1(\"Hello World\")
        )
    )", "app.R")

  tailwindcss <- NULL # can be set to the executable file
  compile_tailwindcss(infile, outfile, tailwindcss = tailwindcss)
  cat(paste(readLines(outfile)[1:20], collapse = "\n"))

  setwd(owd)
}
```

---

install_tailwindcss_cli

Installs the 'TailwindCSS' CLI

Description

This will download the 'TailwindCSS' standalone CLI to the current working directory.

Usage

```
install_tailwindcss_cli(overwrite = FALSE, version = "latest", verbose = FALSE)
```

Arguments

`overwrite`	if existing installations should be overwritten
`version`	the version to install, default is latest
`verbose`	if the version etc should be reported

Details

This will download the 'TailwindCSS' standalone CLI to the current working directory. See here for details on the standalone CLI. This saves you from having to install 'node.js'.

On the mac, after installing the CLI, you need to make sure that the file is executable to run it. For Mac, the easiest way to do so is to ensure you're in the correct working directory in R and type `system("chmod +x tailwindcss")`. Alternatively, you could `cd` to the directory in terminal and then run `chmod +x tailwindcss`.

Value

invisibly the path to the cli program

See Also

compile_tailwindcss

Examples

```
if (interactive()) {
  install_tailwindcss_cli()
}
```

---

is_tailwindcss_installed

Checks if 'TailwindCSS' CLI is installed

Description

To install the CLI of 'TailwindCSS', please follow the instructions of 'TailwindCSS' releases. Make sure that you either provide the direction to the executable as the first argument to this function or put it in a folder on your PATH variable.

Usage

```
is_tailwindcss_installed(tailwindcss = NULL, verbose = FALSE)
```

Arguments

tailwindcss	name and path to the executable
verbose	report version number etc

Value

TRUE/FALSE if the CLI is installed

Examples

```
if (interactive() == TRUE) {
  is_tailwindcss_installed()
}
```

---

twBtnOpenModal *Creates a button to open a Modal Dialog*

Description

Creates a button to open a Modal Dialog

Usage

```
twBtnOpenModal(
  btn_id,
  btn_label,
  btn_class = NULL,
  icon = NULL,
  modal_id = "shiny-modal"
)
```

Arguments

btn_id	ID of the button
btn_label	Label for the button
btn_class	Classes to style the button
icon	an optional icon for the button
modal_id	ID of the modal, make sure that the IDs are identical to the one used in twModalDialog()

Value

a list with a shiny.tag class

Examples

```
ui <- div(
  use_tailwind(),
  class = "h-screen bg-stone-100 p-10",
  twBtnOpenModal(
    "open_modal", "Show Modal",
   btn_class = "px-5 py-2 bg-rose-500 hover:bg-rose-700 text-white cursor-pointer rounded-md"
  ),
  twModalDialog(p("Hello World"), )
)

server <- function(input, output, session) {
  observeEvent(input$open_modal, {
    print("Modal Opened")
  })
  observeEvent(input$submit, {
    print("Modal Closed - Submitted")
  })
  observeEvent(input$close, {
    print("Modal Closed - Closed")
  })
}
if (interactive() == TRUE) shinyApp(ui, server)
```

---

twCheckboxGroupInput *Wrapper around* `shiny::checkboxGroupInput()` *but allowing for more classes*

Description

Wrapper around `shiny::checkboxGroupInput()` but allowing for more classes

Usage

```
twCheckboxGroupInput(
  inputId,
  label,
  choices = NULL,
  selected = NULL,
  inline = FALSE,
  width = NULL,
  container_class = NULL,
  main_label_class = NULL,
  input_class = NULL,
  label_class = NULL,
  inner_container_class = NULL,
  disabled = FALSE
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
choices	List of values to show checkboxes for. If elements of the list are named then that name rather than the value is displayed to the user. If this argument is provided, then choiceNames and choiceValues must not be provided, and vice-versa. The values should be strings; other types (such as logicals and numbers) will be coerced to strings.
selected	The values that should be initially selected, if any.
inline	If TRUE, render the choices inline (i.e. horizontally)
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
container_class	additional classes to be applied to the container
main_label_class	additional classes to be applied to the main label
input_class	additional classes to be applied to the input element
label_class	additional classes to be applied to the label
inner_container_class	additional classes to be applied to the container for each option
disabled	if the user should not be able to interact with the field

Value

a list with a shiny.tag class

See Also

shiny::checkboxGroupInput()

Examples

```
shiny::checkboxGroupInput("id", "label", choices = c("A", "B"))
twCheckboxGroupInput("id", "label",
  choices = c("A", "B"),
  width = "200px", disabled = c(TRUE, FALSE),
  container_class = "OUTER.CONTAINER",
  inner_container_class = c("INNER CONTAINER 1", "INNER CONTAINER 2"),
  label_class = c("LABEL 1", "LABEL 2"),
  input_class = "INPUT-ALL"
)

# basic full shiny example
library(shiny)

ui <- fluidPage(
  use_tailwind(),
  twCheckboxGroupInput(
```

```
    "chks", "Check all that apply:",
    choices = c("This" = "a", "That" = "b", "None (disabled)" = "c"),
    disabled = c(FALSE, FALSE, TRUE),
    container_class = "w-48 m-4 p-2 border border-gray-200 rounded-md drop-shadow-md",
    label_class = "font-serif text-gray-600",
   input_class = "rounded rounded-full text-pink-500 border-pink-200 focus:ring-pink-500",
  ),
  verbatimTextOutput("out")
)

server <- function(input, output) {
  output$out <- renderText({
    input$chks
  })
}

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

`twCheckboxInput` *Wrapper around* `shiny::checkboxInput()` *but allowing for more classes*

Description

Wrapper around `shiny::checkboxInput()` but allowing for more classes

Usage

```
twCheckboxInput(
  inputId,
  label,
  value = FALSE,
  width = NULL,
  disabled = FALSE,
  container_class = NULL,
  label_class = NULL,
  input_class = NULL,
  center = FALSE
)
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`label`	Display label for the control, or `NULL` for no label.
`value`	Initial value (`TRUE` or `FALSE`).
`width`	The width of the input, e.g. `'400px'`, or `'100%'`; see `validateCssUnit()`.
`disabled`	if the user should not be able to interact with the field

container_class
: additional classes to be applied to the container

label_class
: additional classes to be applied to the label

input_class
: additional classes to be applied to the input element

center
: if a margin of 0px !important should be applied, effectively removing bootstrap styling (if applied) to center the checkbox easier

Value

a list with a `shiny.tag` class

See Also

`shiny::checkboxInput()`

Examples

```
shiny::checkboxInput("id", "label", value = FALSE)
twCheckboxInput("id", "label",
  value = TRUE, width = "200px", disabled = TRUE,
  container_class = "CONTAINER", label_class = "LABEL", input_class = "INPUT"
)

# basic full shiny example
library(shiny)

ui <- fluidPage(
  use_tailwind(),
  twCheckboxInput(
    "chk", "Check me!",
    value = TRUE,
    container_class = "w-48 m-4 p-2 border border-gray-200 rounded-md drop-shadow-md",
    label_class = "font-serif text-gray-600",
    input_class = "text-pink-500 focus:ring-pink-500",
    center = TRUE
  ),
  verbatimTextOutput("out")
)

server <- function(input, output) {
  output$out <- renderText({
    input$chk
  })
}

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

twDateInput *Wrapper around* shiny::dateInput() *but allowing for more classes*

Description

Wrapper around shiny::dateInput() but allowing for more classes

Usage

```
twDateInput(
  inputId,
  label,
  value = NULL,
  min = NULL,
  max = NULL,
  format = "yyyy-mm-dd",
  startview = "month",
  weekstart = 0,
  language = "en",
  width = NULL,
  autoclose = TRUE,
  datesdisabled = NULL,
  daysofweekdisabled = NULL,
  container_class = NULL,
  label_class = NULL,
  input_class = NULL,
  label_after_input = FALSE
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
value	The starting date. Either a Date object, or a string in yyyy-mm-dd format. If NULL (the default), will use the current date in the client's time zone.
min	The minimum allowed date. Either a Date object, or a string in yyyy-mm-dd format.
max	The maximum allowed date. Either a Date object, or a string in yyyy-mm-dd format.
format	The format of the date to display in the browser. Defaults to "yyyy-mm-dd".
startview	The date range shown when the input object is first clicked. Can be "month" (the default), "year", or "decade".
weekstart	Which day is the start of the week. Should be an integer from 0 (Sunday) to 6 (Saturday).

language The language used for month and day names. Default is "en". Other valid values include "ar", "az", "bg", "bs", "ca", "cs", "cy", "da", "de", "el", "en-AU", "en-GB", "eo", "es", "et", "eu", "fa", "fi", "fo", "fr-CH", "fr", "gl", "he", "hr", "hu", "hy", "id", "is", "it-CH", "it", "ja", "ka", "kh", "kk", "ko", "kr", "lt", "lv", "me", "mk", "mn", "ms", "nb", "nl-BE", "nl", "no", "pl", "pt-BR", "pt", "ro", "rs-latin", "rs", "ru", "sk", "sl", "sq", "sr-latin", "sr", "sv", "sw", "th", "tr", "uk", "vi", "zh-CN", and "zh-TW".

width The width of the input, e.g. '400px', or '100%'; see validateCssUnit().

autoclose Whether or not to close the datepicker immediately when a date is selected.

datesdisabled Which dates should be disabled. Either a Date object, or a string in yyyy-mm-dd format.

daysofweekdisabled
Days of the week that should be disabled. Should be a integer vector with values from 0 (Sunday) to 6 (Saturday).

container_class
additional classes to be applied to the container

label_class additional classes to be applied to the label

input_class additional classes to be applied to the input element

label_after_input
TRUE/FALSE if the label should be put after the input box. Default is FALSE. Useful for special cases (floating labels), c.f. 04-shiny-inputs example app.

Value

a list with a shiny.tag class

See Also

shiny::dateInput()

Examples

```
shiny::dateInput("date", "A Date")
twDateInput("date", "A Date",
  container_class = "CONTAINER", label_class = "LABEL", input_class = "INPUT"
)

# basic full shiny example
library(shiny)

ui <- fluidPage(
  use_tailwind(),
  twDateInput(
    "date", "A Date",
    # Apply tailwind classes
    container_class = "w-48 m-4 p-2 border border-gray-200 rounded-md drop-shadow-md",
    label_class = "font-mono text-gray-600",
    input_class = "drop-shadow-lg text-gray-600 font-mono rounded-md border-amber-400"
  ),
```

```
  verbatimTextOutput("value")
)

server <- function(input, output) {
  output$value <- renderText({
    as.character(input$date)
  })
}

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

`twDateRangeInput` *Wrapper around* `shiny::dateRangeInput()` *but allowing for more classes*

Description

Wrapper around `shiny::dateRangeInput()` but allowing for more classes

Usage

```
twDateRangeInput(
  inputId,
  label,
  start = NULL,
  end = NULL,
  min = NULL,
  max = NULL,
  format = "yyyy-mm-dd",
  startview = "month",
  weekstart = 0,
  language = "en",
  separator = " to ",
  width = NULL,
  autoclose = TRUE,
  container_class = NULL,
  label_class = NULL,
  input_class = NULL,
  sep_class = NULL,
  label_after_input = FALSE
)
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`label`	Display label for the control, or `NULL` for no label.
`start`	The initial start date. Either a Date object, or a string in `yyyy-mm-dd` format. If NULL (the default), will use the current date in the client's time zone.

end	The initial end date. Either a Date object, or a string in yyyy-mm-dd format. If NULL (the default), will use the current date in the client's time zone.
min	The minimum allowed date. Either a Date object, or a string in yyyy-mm-dd format.
max	The maximum allowed date. Either a Date object, or a string in yyyy-mm-dd format.
format	The format of the date to display in the browser. Defaults to "yyyy-mm-dd".
startview	The date range shown when the input object is first clicked. Can be "month" (the default), "year", or "decade".
weekstart	Which day is the start of the week. Should be an integer from 0 (Sunday) to 6 (Saturday).
language	The language used for month and day names. Default is "en". Other valid values include "ar", "az", "bg", "bs", "ca", "cs", "cy", "da", "de", "el", "en-AU", "en-GB", "eo", "es", "et", "eu", "fa", "fi", "fo", "fr-CH", "fr", "gl", "he", "hr", "hu", "hy", "id", "is", "it-CH", "it", "ja", "ka", "kh", "kk", "ko", "kr", "lt", "lv", "me", "mk", "mn", "ms", "nb", "nl-BE", "nl", "no", "pl", "pt-BR", "pt", "ro", "rs-latin", "rs", "ru", "sk", "sl", "sq", "sr-latin", "sr", "sv", "sw", "th", "tr", "uk", "vi", "zh-CN", and "zh-TW".
separator	String to display between the start and end input boxes.
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
autoclose	Whether or not to close the datepicker immediately when a date is selected.
container_class	additional classes to be applied to the container
label_class	additional classes to be applied to the label
input_class	additional classes to be applied to the input element
sep_class	additional classes to be applied to the separator element
label_after_input	TRUE/FALSE if the label should be put after the input box. Default is FALSE. Useful for special cases (floating labels), c.f. 04-shiny-inputs example app.

Value

a list with a shiny.tag class

See Also

shiny::dateRangeInput()

Examples

```
shiny::dateRangeInput("date", "A Date")
twDateRangeInput(
  "date", "A Date Range",
  container_class = "CONTAINER", label_class = "LABEL",
  input_class = "INPUT", sep_class = "SEP"
```

```
)

# basic full shiny example
library(shiny)

ui <- fluidPage(
  use_tailwind(),
  twDateRangeInput(
    "date", "A Date",
    # Apply tailwind classes
    container_class = "w-48 m-4 p-2 border border-gray-200 rounded-md drop-shadow-md",
    label_class = "font-mono text-gray-600",
    input_class = "drop-shadow-lg text-gray-600 font-mono rounded-md border-amber-400",
    sep_class = "bg-amber-600 text-white font-bold font-mono"
  ),
  verbatimTextOutput("value")
)

server <- function(input, output) {
  output$value <- renderText({
    as.character(input$date)
  })
}

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

`twFileInput` *Wrapper around* `shiny::fileInput()` *but allowing for more classes*

Description

Wrapper around `shiny::fileInput()` but allowing for more classes

Usage

```
twFileInput(
  inputId,
  label,
  multiple = FALSE,
  accept = NULL,
  width = NULL,
  buttonLabel = "Browse...",
  placeholder = "No file selected",
  container_class = NULL,
  label_class = NULL,
  select_class = NULL,
  button_class = NULL,
  progress_class = NULL
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
multiple	Whether the user should be allowed to select and upload multiple files at once. **Does not work on older browsers, including Internet Explorer 9 and earlier.**
accept	A character vector of "unique file type specifiers" which gives the browser a hint as to the type of file the server expects. Many browsers use this prevent the user from selecting an invalid file. A unique file type specifier can be: • A case insensitive extension like .csv or .rds. • A valid MIME type, like text/plain or application/pdf • One of audio/*, video/*, or image/* meaning any audio, video, or image type, respectively.
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
buttonLabel	The label used on the button. Can be text or an HTML tag object.
placeholder	The text to show before a file has been uploaded.
container_class	additional classes to be applied to the container
label_class	additional classes to be applied to the label
select_class	additional classes to be applied to the select elements
button_class	additional classes to be applied to the upload button
progress_class	additional classes to be applied to the progress bar (ie color)

Value

a list with a shiny.tag class

See Also

shiny::fileInput()

Examples

```
shiny::fileInput("id", "label",
  multiple = TRUE, accept = c(".csv", ".rds"),
  width = "200px", buttonLabel = "Upload", placeholder = "Here"
)
twFileInput("id", "label",
  multiple = TRUE, accept = c(".csv", ".rds"),
  width = "200px", buttonLabel = "Upload", placeholder = "Here",
  container_class = "CONTAINER", label_class = "LABEL",
  select_class = "SELECT"
)

# basic full shiny example
library(shiny)
```

```
ui <- fluidPage(
  use_tailwind(),
  twFileInput(
    inputId = "file", label = "Upload", multiple = TRUE,
    buttonLabel = "Upload", placeholder = "Nothing selected",
    container_class = "shadow-md rounded-md bg-gray-50 m-2 p-2 w-96",
    label_class = "font-serif text-red-800",
    select_class = "font-mono font-bold text-red-800 rounded-r-lg",
    button_class = paste(
      "bg-red-800 border-red-800 hover:bg-red-700",
      "hover:border-red-700 text-white hover:text-gray-50"
    ),
    progress_class = "bg-red-800"
  ),
  verbatimTextOutput("data")
)

server <- function(input, output) {
  output$data <- renderText({
    paste(capture.output(str(input$file)), collapse = "\n")
  })
}

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

`twModalDialog` *Creates a Modal Dialog*

Description

Creates a Modal Dialog

Usage

```
twModalDialog(
  ui,
  close_id = "close",
  close_label = "Close",
  close_class = NA,
  submit_id = "submit",
  submit_label = "Submit",
  submit_class = NA,
  title = "Title of Modal",
  modal_id = "shiny-modal",
  modal_width = "max-w-lg"
)
```

Arguments

ui	UI of the modal
close_id	ID for the close button
close_label	Label for the close button, can be a tagList of an icon and the label
close_class	classes for the close button, if NA default values will be used
submit_id	ID for the submit button
submit_label	Label for the submit button, can be a tagList of an icon and the label
submit_class	classes for the submit button, if NA default values will be used
title	title of the modal
modal_id	id of the modal, make sure the ID is identical to the one used in twBtnOpenModal
modal_width	optional class to define the modal width, eg `max-w-4xl` for a wider modal

Value

a list with a `shiny.tag` class

Examples

```
ui <- div(
  use_tailwind(),
  class = "h-screen bg-stone-100 p-10",
  twBtnOpenModal(
    "open_modal", "Show Modal",
   btn_class = "px-5 py-2 bg-rose-500 hover:bg-rose-700 text-white cursor-pointer rounded-md"
  ),
  twModalDialog(p("Hello World"))
)

server <- function(input, output, session) {
  observeEvent(input$open_modal, {
    print("Modal Opened")
  })
  observeEvent(input$submit, {
    print("Modal Closed - Submitted")
  })
  observeEvent(input$close, {
    print("Modal Closed - Closed")
  })
}
if (interactive() == TRUE) shinyApp(ui, server)
```

twNumericInput *Wrapper around* `shiny::numericInput()` *but allowing for more classes*

Description

Wrapper around `shiny::numericInput()` but allowing for more classes

Usage

```
twNumericInput(
  inputId,
  label,
  value,
  min = NA,
  max = NA,
  step = NA,
  width = NULL,
  placeholder = "",
  disabled = FALSE,
  container_class = NULL,
  label_class = NULL,
  input_class = NULL,
  label_after_input = FALSE
)
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`label`	Display label for the control, or `NULL` for no label.
`value`	Initial value.
`min`	Minimum allowed value
`max`	Maximum allowed value
`step`	Interval to use when stepping between min and max
`width`	The width of the input, e.g. `'400px'`, or `'100%'`; see `validateCssUnit()`.
`placeholder`	Placeholder text for numeric input. Disappears after input
`disabled`	if the user should not be able to interact with the field
`container_class`	additional classes to be applied to the container
`label_class`	additional classes to be applied to the label
`input_class`	additional classes to be applied to the input element
`label_after_input`	TRUE/FALSE if the label should be put after the input box. Default is FALSE. Useful for special cases (floating labels), c.f. 04-shiny-inputs example app.

Value

a list with a shiny.tag class

See Also

shiny::numericInput()

Examples

```
shiny::numericInput("number", "A Number", 42, min = 10, max = 100, step = 13, width = "200px")
twNumericInput("number", "A Number", 42,
  min = 10, max = 100, step = 13, width = "200px",
  container_class = "CONTAINER", label_class = "LABEL", input_class = "INPUT"
)

# basic full shiny example
library(shiny)

ui <- fluidPage(
  use_tailwind(),
  twNumericInput(
    "number", "A Number", 123456,
    # Apply tailwind classes
    container_class = "w-48 m-4 p-2 border border-gray-200 rounded-md drop-shadow-md",
    label_class = "font-mono text-gray-600",
    input_class = "drop-shadow-lg text-gray-600 font-mono rounded-md border-amber-400"
  ),
  verbatimTextOutput("value")
)

server <- function(input, output) {
  output$value <- renderText({
    input$number
  })
}

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

---

twSelectInput *Wrapper around shiny::selectInput() but allowing for more classes*

Description

Wrapper around shiny::selectInput() but allowing for more classes

Usage

```
twSelectInput(
  inputId,
  label,
  choices,
  selected = NULL,
  multiple = FALSE,
  selectize = TRUE,
  width = NULL,
  size = NULL,
  container_class = NULL,
  label_class = NULL,
  select_class = NULL
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
choices	List of values to select from. If elements of the list are named, then that name — rather than the value — is displayed to the user. It's also possible to group related inputs by providing a named list whose elements are (either named or unnamed) lists, vectors, or factors. In this case, the outermost names will be used as the group labels (leveraging the <optgroup> HTML tag) for the elements in the respective sublist. See the example section for a small demo of this feature.
selected	The initially selected value (or multiple values if multiple = TRUE). If not specified then defaults to the first value for single-select lists and no values for multiple select lists.
multiple	Is selection of multiple items allowed?
selectize	Whether to use **selectize.js** or not.
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
size	Number of items to show in the selection box; a larger number will result in a taller box. Not compatible with selectize=TRUE. Normally, when multiple=FALSE, a select input will be a drop-down list, but when size is set, it will be a box instead.
container_class	additional classes to be applied to the container
label_class	additional classes to be applied to the label
select_class	additional classes to be applied to the select elements

Value

a list with a shiny.tag class

See Also

shiny::selectInput()

Examples

```
shiny::selectInput("id", "label", c("A" = "a", "B" = "b", "C" = "c"),
  selected = c("a", "b"), width = "200px",
  multiple = TRUE
)
twSelectInput("id", "label", c("A" = "a", "B" = "b", "C" = "c"),
  selected = c("a", "b"), width = "200px",
  multiple = TRUE, selectize = TRUE,
  container_class = "CONTAINER", label_class = "LABEL",
  select_class = "SELECT"
)

# basic full shiny example
library(shiny)

ui <- fluidPage(
  use_tailwind(),
  twSelectInput(
    "variable", "Variable to select:",
    c("Cylinders" = "cyl", "Transmission" = "am", "Gears" = "gear"),
    multiple = TRUE,
    # Apply tailwind classes
    container_class = "shadow-md rounded-md bg-gray-50 m-4 p-2 w-72",
    label_class = "font-serif",
    select_class = "font-mono font-bold text-red-800 rounded-md bg-stone-50"
  ),
  tableOutput("data")
)

server <- function(input, output) {
  output$data <- renderTable(
    {
      mtcars[, c("mpg", input$variable), drop = FALSE]
    },
    rownames = TRUE
  )
}

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

`twSelectizeInput` *Wrapper around* `shiny::selectizeInput()` *but allowing for more classes*

Description

Note that the colors for the slider bar can be customized by overriding the `irs` class. c.f. 05-apply-directive example app

Usage

```
twSelectizeInput(
  inputId,
  ...,
  options = NULL,
  width = NULL,
  container_class = NULL,
  label_class = NULL,
  input_class = NULL,
  label_after_input = FALSE
)
```

Arguments

inputId	The input slot that will be used to access the value.
...	Arguments passed to selectInput().
options	A list of options. See the documentation of **selectize.js** for possible options (character option values inside base::I() will be treated as literal JavaScript code; see renderDataTable() for details).
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
container_class	additional classes to be applied to the container
label_class	additional classes to be applied to the label
input_class	additional classes to be applied to the input element
label_after_input	TRUE/FALSE if the label should be put after the input box. Default is FALSE. Useful for special cases (floating labels), c.f. 05-apply-directive example app.

Value

a list with a shiny.tag class

See Also

shiny::selectizeInput()

Examples

```
shiny::selectizeInput("selectize", "A Selection", choice = c("A", "B"))
twSelectizeInput("selectize", "A Selection",
  choice = c("A", "B"),
  container_class = "CONTAINER", label_class = "LABEL",
  input_class = "INPUT"
)

# basic full shiny example
library(shiny)
```

```
ui <- fluidPage(
  use_tailwind(),
  twSelectizeInput(
    "values", "A Selection",
    choice = c("A", "B"), multiple = TRUE,
    # Apply tailwind classes
    container_class = "w-48 m-4 p-2 border border-gray-200 rounded-md drop-shadow-md",
    label_class = "font-mono text-gray-600",
    input_class = "drop-shadow-lg text-gray-600 font-mono rounded-md border-amber-400"
  ),
  verbatimTextOutput("value")
)

server <- function(input, output) {
  output$value <- renderText({
    as.character(input$values)
  })
}

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

`twSliderInput` *Wrapper around* `shiny::sliderInput()` *but allowing for more classes*

Description

Note that the colors for the slider bar can be customized by overriding the `irs` class. c.f. 05-apply-directive example app

Usage

```
twSliderInput(
  inputId,
  label,
  min,
  max,
  value,
  step = NULL,
  round = FALSE,
  ticks = TRUE,
  animate = FALSE,
  width = NULL,
  sep = ",",
  pre = NULL,
  post = NULL,
  timeFormat = NULL,
  timezone = NULL,
  dragRange = TRUE,
```

```
  container_class = NULL,
  label_class = NULL,
  input_class = NULL,
  label_after_input = FALSE
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
min, max	The minimum and maximum values (inclusive) that can be selected.
value	The initial value of the slider, either a number, a date (class Date), or a date-time (class POSIXt). A length one vector will create a regular slider; a length two vector will create a double-ended range slider. Must lie between min and max.
step	Specifies the interval between each selectable value on the slider. Either NULL, the default, which uses a heuristic to determine the step size or a single number. If the values are dates, step is in days; if the values are date-times, step is in seconds.
round	TRUE to round all values to the nearest integer; FALSE if no rounding is desired; or an integer to round to that number of digits (for example, 1 will round to the nearest 10, and -2 will round to the nearest .01). Any rounding will be applied after snapping to the nearest step.
ticks	FALSE to hide tick marks, TRUE to show them according to some simple heuristics.
animate	TRUE to show simple animation controls with default settings; FALSE not to; or a custom settings list, such as those created using animationOptions().
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
sep	Separator between thousands places in numbers.
pre	A prefix string to put in front of the value.
post	A suffix string to put after the value.
timeFormat	Only used if the values are Date or POSIXt objects. A time format string, to be passed to the Javascript strftime library. See https://github.com/samsonjs/strftime for more details. The allowed format specifications are very similar, but not identical, to those for R's base::strftime() function. For Dates, the default is "%F" (like "2015-07-01"), and for POSIXt, the default is "%F %T" (like "2015-07-01 15:32:10").
timezone	Only used if the values are POSIXt objects. A string specifying the time zone offset for the displayed times, in the format "+HHMM" or "-HHMM". If NULL (the default), times will be displayed in the browser's time zone. The value "+0000" will result in UTC time.
dragRange	This option is used only if it is a range slider (with two values). If TRUE (the default), the range can be dragged. In other words, the min and max can be dragged together. If FALSE, the range cannot be dragged.
container_class	additional classes to be applied to the container

label_class additional classes to be applied to the label

input_class additional classes to be applied to the input element

label_after_input
TRUE/FALSE if the label should be put after the input box. Default is FALSE. Useful for special cases (floating labels), c.f. 05-apply-directive example app.

Value

a list with a shiny.tag class

See Also

shiny::sliderInput()

Examples

```
shiny::sliderInput("values", "A Range", min = 0, max = 100, value = 75)
twSliderInput("values", "A Range",
  min = 0, max = 100, value = 75,
  container_class = "CONTAINER", label_class = "LABEL",
  input_class = "INPUT"
)

# basic full shiny example
library(shiny)

ui <- fluidPage(
  use_tailwind(),
  twSliderInput(
    "values", "A Range",
    min = 0, max = 100, value = 75,
    # Apply tailwind classes
    container_class = "w-48 m-4 p-2 border border-gray-200 rounded-md drop-shadow-md",
    label_class = "font-mono text-gray-600",
    input_class = "drop-shadow-lg text-gray-600 font-mono rounded-md border-amber-400"
  ),
  verbatimTextOutput("value")
)

server <- function(input, output) {
  output$value <- renderText({
    as.character(input$date)
  })
}

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

twTabContent *Creates the Content Elements of Tabs*

Description

This function only creates the content elements of tabs, the navigation elements can by created by the `twTabNav()` function. A full example is included in the example 06-sidebar-dashboard.

Usage

```
twTabContent(
  ...,
  ids = NULL,
  container_class = NULL,
  content_class = NULL,
  tabsetid = "tabSet1"
)
```

Arguments

`...`	UI element to include in the tab
`ids`	a list of reference IDs to the navigation elements. This will be overridden by ID fields of the `...` values (if given). Default is `twTab-{i}-content` (note that the ids must end with `-content` where the part before matches the IDs of the navigation elements. Note that this option is only needed when multiple tab systems are used within a page or when the elements of the `twTabContents` are given out of order.
`container_class`	additional classes to be applied to the container
`content_class`	additional classes to be applied to each content container
`tabsetid`	an optional class that is added to the container to be identify and linked the tabsets. Must match the `tabsetid` of `twTabContent()`. Can be an arbitrary text, but due to it being a class, make sure to not have class-clashes (eg `"button"` would be a bad idea). This allows to have multiple nested tabsets. See also Example 09-nested-tabsets.

Details

Note that contrary how `shiny::tabPanel()` constructs a tab page, these funtions (`twTabContent()` and `twTabNav()`) construct navigation and content independently, allowing more flexibility.

The active elements all have either a `twTab-active` or `twTabContent-active` CSS class if their styling needs to be overriden (see also the example).

Value

a list with a `shiny.tag` class

See Also

twTabNav()

Examples

```
twTabContent(
  div(h1("First Tab"), shiny::plotOutput("plot1")),
  div(h1("Second Tab"), shiny::plotOutput("plot2"))
)

if (interactive()) {
################################################################################
# Example App

library(shiny)
# basic Tabs

ui_basic <- shiny::div(
  shiny::h1("Completely Unstyled Tabs..."),
  twTabNav(
    shiny::div("Tab 1 (click me)"),
    shiny::div("Tab 2 (click me)")
  ),
  twTabContent(
    shiny::div(shiny::h1("First Tab"), shiny::plotOutput("plot1")),
    shiny::div(shiny::h1("Second Tab"), shiny::plotOutput("plot2"))
  )
)

server <- function(input, output, session) {
  output$plot1 <- shiny::renderPlot({
    print("Plot 1")
    plot(1:10, rnorm(10))
  })
  output$plot2 <- shiny::renderPlot({
    print("Plot 2")
    plot(1:100, rnorm(100))
  })
}

if (interactive()) shiny::shinyApp(ui_basic, server)

################################################################################
# Styled App

ui_styled <- shiny::div(
  class = "h-screen bg-white overflow-hidden flex",
  shiny.tailwind::use_tailwind(),
  twTabNav(
    shiny::div(icon("database"), shiny::span("Tab One", class = "pl-2")),
    shiny::div(icon("server"), shiny::span("Tab Two", class = "pl-2")),
    container_class = "h-full pt-10 pt-2 bg-indigo-900",
    tab_class = "cursor-pointer py-2 px-4 my-4 w-full text-white hover:bg-indigo-700"
```

```
  ),
  twTabContent(
    shiny::div(
      shiny::h1("First Tab",
        class = "p-10 text-center font-sans text-8xl font-extrabold text-slate-800"
      ),
      shiny::plotOutput("plot1")
    ),
    shiny::div(
      shiny::h1("Second Tab",
        class = "p-10 text-center font-sans text-8xl font-extrabold text-slate-800"
      ),
      shiny::plotOutput("plot2")
    ),
    container_class = "flex-1 bg-indigo-50"
  )
)

if (interactive()) shiny::shinyApp(ui_styled, server)
```

---

twTabNav *Creates the Navigation Element of Tabs*

Description

This function creates only the navigation elements of tabs, the content elements can be created by the `twTabContent()` function. A full example is included in the example 06-sidebar-dashboard.

Usage

```
twTabNav(
  ...,
  ids = NULL,
  container_class = NULL,
  tab_class = NULL,
  tabsetid = "tabSet1"
)
```

Arguments

`...`	titles for the navigation elements
`ids`	a list of reference IDs for each tab. This will be overridden by ID fields of the `...` values (if given). Default is `twTab-{i}`. Note that this option is only needed when multiple tab systems are used within a page or when the elements of the `twTabContents` are given out of order.
`container_class`	additional classes to be applied to the container

tab_class	additional classes to be applied to each tab container
tabsetid	an optional class that is added to the container to be identify and linked the tabsets. Must match the tabsetid of twTabContent(). Can be an arbitrary text, but due to it being a class, make sure to not have class-clashes (eg "button" would be a bad idea). This allows to have multiple nested tabsets. See also Example 09-nested-tabsets.

Details

Note that contrary how shiny::tabPanel() constructs a tab page, these funtions (twTabContent() and twTabNav()) construct navigation and content independently, allowing more flexibility.

The active elements all have either a twTab-active or twTabContent-active CSS class if their styling needs to be overriden (see also the example).

Value

a list with a shiny.tag class

See Also

twTabContent()

Examples

```
twTabNav(
  div("Tab 1", id = "firstTab"),
  div("Tab 2", id = "secondTab"),
  container_class = "CONTAINER", tab_class = "TAB"
)

################################################################################
# Example App

library(shiny)
# basic Tabs

ui_basic <- shiny::div(
  shiny::h1("Completely Unstyled Tabs..."),
  twTabNav(
    shiny::div("Tab 1 (click me)"),
    shiny::div("Tab 2 (click me)")
  ),
  twTabContent(
    shiny::div(shiny::h1("First Tab"), shiny::plotOutput("plot1")),
    shiny::div(shiny::h1("Second Tab"), shiny::plotOutput("plot2"))
  )
)

server <- function(input, output, session) {
  output$plot1 <- shiny::renderPlot({
    print("Plot 1")
```

```
    plot(1:10, rnorm(10))
  })
  output$plot2 <- shiny::renderPlot({
    print("Plot 2")
    plot(1:100, rnorm(100))
  })
}

if (interactive()) shiny::shinyApp(ui_basic, server)

################################################################################
# Styled App

ui_styled <- div(
  class = "h-screen bg-white overflow-hidden flex",
  shiny.tailwind::use_tailwind(),
  twTabNav(
    div(icon("database"), span("Tab One", class = "pl-2")),
    div(icon("server"), span("Tab Two", class = "pl-2")),
    container_class = "h-full pt-10 pt-2 bg-indigo-900",
    tab_class = "cursor-pointer py-2 px-4 my-4 w-full text-white hover:bg-indigo-700"
  ),
  twTabContent(
    div(
      h1("First Tab",
        class = "p-10 text-center font-sans text-8xl font-extrabold text-slate-800"
      ),
      plotOutput("plot1")
    ),
    div(
      h1("Second Tab",
        class = "p-10 text-center font-sans text-8xl font-extrabold text-slate-800"
      ),
      plotOutput("plot2")
    ),
    container_class = "flex-1 bg-indigo-50"
  )
)

if (interactive()) shiny::shinyApp(ui_styled, server)
```

`twTextAreaInput` *Wrapper around* `shiny::textAreaInput()` *but allowing for more classes*

Description

Wrapper around `shiny::textAreaInput()` but allowing for more classes

Usage

```
twTextAreaInput(
  inputId,
  label,
  value = "",
  placeholder = NULL,
  width = NULL,
  height = NULL,
  rows = NULL,
  cols = NULL,
  resize = NULL,
  container_class = NULL,
  label_class = NULL,
  input_class = NULL,
  label_after_input = FALSE
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
value	Initial value.
placeholder	A character string giving the user a hint as to what can be entered into the control. Internet Explorer 8 and 9 do not support this option.
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
height	The height of the input, e.g. '400px', or '100%'; see validateCssUnit().
rows	The value of the visible character rows of the input, e.g. 6. If the height argument is specified, height will take precedence in the browser's rendering.
cols	Value of the visible character columns of the input, e.g. 80. This argument will only take effect if there is not a CSS width rule defined for this element; such a rule could come from the width argument of this function or from a containing page layout such as fluidPage().
resize	Which directions the textarea box can be resized. Can be one of "both", "none", "vertical", and "horizontal". The default, NULL, will use the client browser's default setting for resizing textareas.
container_class	additional classes to be applied to the container
label_class	additional classes to be applied to the label
input_class	additional classes to be applied to the input element
label_after_input	TRUE/FALSE if the label should be put after the input box. Default is FALSE. Useful for special cases (floating labels), c.f. 04-shiny-inputs example app.

Value

a list with a shiny.tag class

See Also

shiny::textAreaInput()

Examples

```
shiny::textAreaInput("id", "Label",
  value = "The value", width = "200px",
  placeholder = "Placeholder"
)
twTextAreaInput("id", "Label",
  value = "The value", width = "200px",
  height = "200px", placeholder = "Placeholder",
  container_class = "CONTAINER", label_class = "LABEL", input_class = "INPUT"
)

# basic full shiny example
library(shiny)

ui <- fluidPage(
  use_tailwind(),
  twTextAreaInput(
    "text", "A Text",
    placeholder = "Here goes a placeholder",
    width = "400px", height = "400px",
    # Apply tailwind classes
    container_class = "w-48 m-4 p-2 border border-gray-200 rounded-md drop-shadow-md",
    label_class = "font-serif text-gray-600",
    input_class = "drop-shadow-lg font-mono text-gray-600 rounded-md border-amber-400"
  ),
  verbatimTextOutput("value")
)

server <- function(input, output) {
  output$value <- renderText(input$text)
}

if (interactive()) shiny::shinyApp(ui_basic, server)
```

---

twTextInput *Wrapper around shiny::textInput() but allowing for more classes*

Description

Wrapper around shiny::textInput() but allowing for more classes

Usage

```
twTextInput(
  inputId,
```

```
  label = NULL,
  value = NULL,
  placeholder = NULL,
  width = NULL,
  type = "text",
  container_class = NULL,
  label_class = NULL,
  input_class = NULL,
  label_after_input = FALSE
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
value	Initial value.
placeholder	A character string giving the user a hint as to what can be entered into the control. Internet Explorer 8 and 9 do not support this option.
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
type	the type for the input, eg "text" (default), "password", "email", "month", "url", ... see also MDN Input Types
container_class	additional classes to be applied to the container
label_class	additional classes to be applied to the label
input_class	additional classes to be applied to the input element
label_after_input	TRUE/FALSE if the label should be put after the input box. Default is FALSE. Useful for special cases (floating labels), c.f. 04-shiny-inputs example app.

Value

a list with a shiny.tag class

See Also

shiny::textInput()

Examples

```
shiny::textInput(
  "id", "Label",
  value = "The value", width = "200px",
  placeholder = "Placeholder"
)
twTextInput(
  "id", "Label",
  value = "The value", width = "200px",
```

```
  placeholder = "Placeholder", type = "email",
  container_class = "CONTAINER", label_class = "LABEL",
  input_class = "INPUT"
)

# basic full shiny example
library(shiny)
# basic example
ui <- fluidPage(
  use_tailwind(),
  div(
    class = "flex flex-wrap",
    twTextInput(
      "text", "A Text",
      type = "text", placeholder = "Some Text",
      # Apply tailwind classes
      container_class = paste(
        "w-48 m-4 p-2 border border-gray-200",
        "rounded-md drop-shadow-md"
      ),
      label_class = "font-serif text-gray-600",
      input_class = paste(
        "drop-shadow-lg font-mono text-gray-600",
        "rounded-md border-amber-400"
      )
    ),
    twTextInput(
      "email", "An Email",
      type = "email",
      placeholder = "email",
      # Apply tailwind classes
      container_class = paste(
        "w-48 m-4 p-2 border border-gray-200",
        "rounded-md drop-shadow-md"
      ),
      label_class = "font-serif text-gray-600",
      input_class = paste(
        "drop-shadow-lg font-mono text-gray-600",
        "rounded-md border-amber-400"
      )
    ),
    twTextInput(
      "pw", "A Password",
      type = "password",
      placeholder = "dont let it be password",
      # Apply tailwind classes
      container_class = paste(
        "w-48 m-4 p-2 border border-gray-200",
        "rounded-md drop-shadow-md"
      ),
      label_class = "font-serif text-gray-600",
      input_class = paste(
        "drop-shadow-lg font-mono text-gray-600",
```

```
        "rounded-md border-amber-400"
      )
    )
  ),
  twTextInput(
    "pw", "A Password",
    type = "password", placeholder = "dont let it be password",
    # Apply tailwind classes
    container_class = "w-48 m-4 p-2 border border-gray-200 rounded-md drop-shadow-md",
    label_class = "font-serif text-gray-600",
    input_class = "drop-shadow-lg font-mono text-gray-600 rounded-md border-amber-400"
  ),
  verbatimTextOutput("value")
)

server <- function(input, output) {
  output$value <- renderText({
    paste(capture.output(str(list(
      text = input$text,
      email = input$email,
      pw = input$pw
    ))), collapse = "\n")
  })
}

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

`twVarSelectInput` *Wrapper around* `shiny::varSelectInput()` *but allowing for more classes*

Description

Wrapper around `shiny::varSelectInput()` but allowing for more classes

Usage

```
twVarSelectInput(
  inputId,
  label,
  data,
  selected = NULL,
  multiple = FALSE,
  selectize = TRUE,
  width = NULL,
  container_class = NULL,
  label_class = NULL,
  select_class = NULL
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
data	A data frame. Used to retrieve the column names as choices for a selectInput()
selected	The initially selected value (or multiple values if multiple = TRUE). If not specified then defaults to the first value for single-select lists and no values for multiple select lists.
multiple	Is selection of multiple items allowed?
selectize	Whether to use **selectize.js** or not.
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
container_class	additional classes to be applied to the container
label_class	additional classes to be applied to the label
select_class	additional classes to be applied to the select elements

Value

a list with a shiny.tag class

See Also

shiny::varSelectInput()

Examples

```
shiny::varSelectInput("id", "label", mtcars,
  width = "200px",
  selected = c("vs", "cyl"), multiple = TRUE
)
twVarSelectInput("id", "label", mtcars,
  selected = c("vs", "cyl"), width = "200px",
  multiple = TRUE, selectize = TRUE,
  container_class = "CONTAINER", label_class = "LABEL",
  select_class = "SELECT"
)

# basic full shiny example
library(shiny)
# basic example
ui <- fluidPage(
  use_tailwind(),
  twVarSelectInput(
    "variable", "Variable to select:",
    mtcars,
    multiple = TRUE,
    # Apply tailwind classes
    container_class = "shadow-md rounded-md bg-gray-50 m-4 p-2 w-64",
    label_class = "font-serif",
```

```
    select_class = "font-mono font-bold text-red-800 rounded-md bg-stone-50"
  ),
  tableOutput("data")
)

server <- function(input, output) {
  output$data <- renderTable(
    {
      mtcars[[input$variable]]
    },
    rownames = TRUE
  )
}

if (interactive()) shiny::shinyApp(ui_basic, server)
```

`twVarSelectizeInput` *Wrapper around* `shiny::varSelectizeInput()` *but allowing for more classes*

Description

Note that the colors for the selected elements can be customized. c.f. 05-apply-directive example app

Usage

```
twVarSelectizeInput(
  inputId,
  ...,
  options = NULL,
  width = NULL,
  container_class = NULL,
  label_class = NULL,
  input_class = NULL,
  label_after_input = FALSE
)
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`...`	Arguments passed to `varSelectInput()`.
`options`	A list of options. See the documentation of **selectize.js** for possible options (character option values inside `base::I()` will be treated as literal JavaScript code; see `renderDataTable()` for details).
`width`	The width of the input, e.g. `'400px'`, or `'100%'`; see `validateCssUnit()`.

container_class
: additional classes to be applied to the container

label_class
: additional classes to be applied to the label

input_class
: additional classes to be applied to the input element

label_after_input
: TRUE/FALSE if the label should be put after the input box. Default is FALSE. Useful for special cases (floating labels), c.f. 05-apply-directive example app.

Value

a list with a `shiny.tag` class

See Also

`shiny::varSelectizeInput()`

Examples

```
shiny::varSelectizeInput("selectize", "A Selection", mtcars)
twVarSelectizeInput("selectize", "A Selection", mtcars,
  container_class = "CONTAINER", label_class = "LABEL",
  input_class = "INPUT"
)

# basic full shiny example
library(shiny)

ui <- fluidPage(
  use_tailwind(),
  twVarSelectizeInput(
    "values", "A Selection", mtcars,
    multiple = TRUE,
    # Apply tailwind classes
    container_class = "w-48 m-4 p-2 border border-gray-200 rounded-md drop-shadow-md",
    label_class = "font-mono text-gray-600",
    input_class = "drop-shadow-lg text-gray-600 font-mono rounded-md border-amber-400"
  ),
  verbatimTextOutput("value")
)

server <- function(input, output) {
  output$value <- renderText({
    as.character(input$values)
  })
}

if (interactive()) shiny::shinyApp(ui_basic, server)
```

use_daisyui *Allows you to use 'daisyUI' elements*

Description

See also: https://daisyui.com/ and https://daisyui.com/components/

Usage

```
use_daisyui(version = "2.17.0", ...)
```

Arguments

version	the version of 'daisyUI' to use, default is 2.17.0
...	additional arguments passed to use_tailwind()

Details

Note that this uses the CDN version, which is not recommended for production by 'daisyUI'.

Value

the required HTML-head tags to use 'daisyUI' as shiny.tag

Examples

```
library(shiny)

ui <- div(
  class = "h-full w-full",
  use_daisyui(),
  div(
    class = "text-sm breadcrumbs",
    tags$ul(
      tags$li(tags$a("Home")),
      tags$li(tags$a("Documents")),
      tags$li(tags$a("Add Documents"))
    )
  )
)
if (interactive()) shiny::shinyApp(ui, function(input, output) {})
```

use_flowbite *Allows you to use 'flowbite' components*

Description

See also: https://flowbite.com/ and https://flowbite.com/#components

Usage

```
use_flowbite(version = "1.4.7", ...)
```

Arguments

version	the version of 'flowbite' to use, default is 1.4.7
...	further arguments passed to use_tailwind()

Value

the required HTML-head tags to use 'flowbite' as shiny.tag

use_tailwind *'TailwindCSS' with Shiny*

Description

'TailwindCSS' with Shiny

Usage

```
use_tailwind(css = NULL, tailwindConfig = NULL)
```

Arguments

css	Optional. Path to ".css" file. Can use @apply tags for applying Tailwind classes. See description for more details.
tailwindConfig	Optional. Path to ".js" file containing json object tailwind.config = {}. See https://github.com/tailwindlabs/tailwindcss/releases/tag/v3.0.0-alpha.1.

Details

'TailwindCSS' is a utility-based design framework that makes designing simple. See details in the README for this package for why this is so great.

However, the complete set of tailwind css classes is massive (~15mb), so you don't want to load all of these. That is where Tailwind's new Just in Time compiling comes in. It will only load the css classes you use, as you use them. So if your shiny app renders ui dynamically, it will load appropriate css whenever the UI is rendered.

Custom css can use the `@apply` directives that come with tailwind to easily compile set of classes. See https://tailwindcss.com/docs/functions-and-directives#apply for more details. It just *has* to be passed to the use_tailwind function if you want to use the `@apply` directive.

Custom configuration of tailwind is also possible. There are two options available in `use_tailwind`. First, if you don't want to use any custom modules, uses tailwindConfig. An example is in the folder `inst/examples/02-config` in the github repository. Note the `.js` file should only consist of the creation of the JSON object `tailwind.config = {}`. The function will place it in the appropriate script tag.

Value

a list of type `shiny.tag` with head and script elements needed to run a tailwind app

Examples

```
library(shiny)
example_apps <- list.files(system.file("examples", package = "shiny.tailwind"),
  full.names = TRUE
)
basename(example_apps)

if (interactive()) runApp(example_apps[1])
```

Index