Package 'shidashi'

November 17, 2021

Type Package

Title A Shiny Dashboard Template System

Version 0.1.0

Language en-US

URL https://dipterix.org/shidashi/
https://github.com/dipterix/shidashi

BugReports https://github.com/dipterix/shidashi/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.1.2

Description A template system based on 'AdminLTE3'
(<https://adminlte.io/themes/v3/>)
theme. Comes with default theme that can be easily customized.
Developers can upload modified templates on 'Github', and users can
easily download templates with 'RStudio' project wizard.
The key features of the default template include light and dark theme
switcher, resizing graphs, synchronizing inputs across sessions,
new notification system, fancy progress bars, and card-like flip
panels with back sides, as well as various of 'HTML' tool widgets.

Imports digest (>= 0.6.27), fastmap (>= 1.1.0), formatR (>= 1.11),
httr (>= 1.4.2), shiny (>= 1.7.0), yaml (>= 2.2.1), jsonlite
(>= 1.7.2)

Suggests htmltools (>= 0.5.2), logger (>= 0.2.1), rstudioapi (>=
0.13), ggplot2, ggExtra

NeedsCompilation no

Author Zhengjia Wang [cph, aut, cre] (<https://orcid.org/0000-0001-5629-1116>),
ColorlibHQ [cph] (AdminLTE - Bootstrap 4 Admin Dashboard),
Bootstrap contributors [ctb] (Bootstrap library),
Twitter, Inc [cph] (Bootstrap library),
Ivan Sagalaev [ctb, cph] (highlight.js library),
Rene Haas [ctb, cph] (OverlayScrollbars library),
Zeno Rocha [ctb, cph] (Clipboard.js library)

Maintainer Zhengjia Wang <dipterix.wang@gmail.com>

Repository CRAN

Date/Publication 2021-11-17 21:20:02 UTC

R topics documented:

`accordion` *Generates an 'accordion' tab-set*

Description

Generates an 'accordion' tab-set that only one tab is expanded at a time. This feature is experimental and has bugs in some situations. Please use it at your own risk.

Usage

```
accordion(
  ...,
  id = rand_string(),
  class = NULL,
  style_header = NULL,
  style_body = NULL,
  env = parent.frame(),
  extras = list(),
  root_path = template_root()
)
```

Arguments

...	'accordion' items, generated by accordion_item
id	the element id, must be unique
class	the additional 'HTML' class
style_header	additional 'CSS' styles for header
style_body	additional 'CSS' styles for content body
env	environment to evaluate . . .
extras	key-value pairs that overrides the parameters in accordion_item
root_path	see template_root

Value

'shiny.tag.list' 'HTML' tags

See Also

accordion_item

Examples

```
library(shiny)
library(shidashi)

accordion(
  id = "input-set",
  accordion_item(
    title = "Input Group A",
    textInput("input_1", "Input 1"),
    collapsed = FALSE,
    footer = "Anim pariatur cliche reprehenderit dolor brunch.",
    tools = list(
      as_badge("New|badge-danger")
      # card_tool(widget = "collapse")
    )
```

```
  ),
  accordion_item(
    title = "Input Group B",
    textInput("input_2", "Input 2"),
    footer = actionButton("btn1", "OK"),
    collapsed = FALSE,
    tools = list(
      card_tool(widget = "link",
                icon = shiny::icon("question-circle"),
                href = "#")
    )
  )
)
```

`accordion_item` *'Accordion' items*

Description

'Accordion' items

Usage

```
accordion_item(
  title,
  ...,
  footer = NULL,
  tools = NULL,
  class = "",
  collapsed = TRUE,
  parentId = rand_string(),
  itemId = rand_string(),
  style_header = NULL,
  style_body = NULL,
  root_path = template_root()
)
```

Arguments

`title`	character title to show in the header
`...`	body content
`footer`	footer element, hidden if `NULL`
`tools`	a list of badge or tool icons generated by `card_tool` or `as_badge`
`class`	the class of the item
`collapsed`	whether collapsed at the beginning
`parentId`	parent `accordion` id

itemId	the item id
style_header, style_body	'CSS' style of item header and body
root_path	see template_root

Value

'shiny.tag.list' 'HTML' tags

See Also

accordion

---

adminlte *Generates 'AdminLTE' theme-related 'HTML' tags*

Description

These functions should be called in 'HTML' templates. Please see vignettes for details.

Usage

```
adminlte_ui(root_path = template_root())

adminlte_sidebar(
  root_path = template_root(),
  settings_file = "modules.yaml",
  shared_id = rand_string(26)
)
```

Arguments

root_path	the root path of the website project; see template_settings
settings_file	the settings file containing the module information
shared_id	a shared identification by session to synchronize the inputs; assigned internally.

Value

'HTML' tags

as_badge *Generates badge icons*

Description

Usually used along with `card`, `card2`, and `card_tabset`. See `tools` parameters in these functions accordingly.

Usage

```
as_badge(badge = NULL)
```

Arguments

badge	characters, `"shiny.tag"` object or `NULL`

Details

When `badge` is `NULL` or empty, then `as_badge` returns empty strings. When `badge` is a `"shiny.tag"` object, then 'HTML' class `'right'` and `'badge'` will be appended. When `badge` is a string, it should follow the syntax of `"message|class"`. The text before `"|"` will be the badge message, and the text after the `"|"` becomes the class string.

Value

'HTML' tags

Examples

```
# Basic usage
as_badge("New")

# Add class `bg-red` and `no-padding`
as_badge("New|bg-red no-padding")
```

as_icon *Convert characters, shiny icons into 'fontawesome' 4*

Description

Convert characters, shiny icons into 'fontawesome' 4

Usage

```
as_icon(icon = NULL, class = "fas")
```

Arguments

icon	character or icon
class	icon class; change this when you are using 'fontawesome' professional version. The choices are 'fa' (compatible), 'fas' (strong), 'far' (regular), 'fal' (light), and 'fad' (duo-tone).

Value

'HTML' tag

Examples

```
if(interactive()){
as_icon("bookmark", class = "far")
as_icon("bookmark", class = "fas")

# no icon
as_icon(NULL)
}
```

---

back_top_button *'HTML' code to generate small back-to-top button*

Description

This function is a template function that should be called in 'HTML' templates before closing the "</body>" tag.

Usage

```
back_top_button(icon = "chevron-up", title = "Jump to")
```

Arguments

icon	the icon for back-to-top button
title	the expanded menu title

Value

'HTML' tags

Examples

```
back_top_button()
back_top_button("rocket")
```

`card` *Card-like 'HTML' element*

Description

Card-like 'HTML' element

Usage

```
card(
  title,
  ...,
  footer = NULL,
  tools = NULL,
  inputId = NULL,
  class = "",
  class_header = "",
  class_body = "",
  class_foot = "",
  style_header = NULL,
  style_body = NULL,
  start_collapsed = FALSE,
  resizable = FALSE,
  root_path = template_root()
)

card2(
  title,
  body_main,
  body_side = NULL,
  footer = NULL,
  tools = NULL,
  inputId = NULL,
  class = "",
  class_header = "",
  class_body = "min-height-400",
  class_foot = "",
  style_header = NULL,
  style_body = NULL,
  start_collapsed = FALSE,
  root_path = template_root()
```

```
)

card2_open(inputId, session = shiny::getDefaultReactiveDomain())

card2_close(inputId, session = shiny::getDefaultReactiveDomain())

card2_toggle(inputId, session = shiny::getDefaultReactiveDomain())

card_operate(inputId, method, session = shiny::getDefaultReactiveDomain())
```

Arguments

`title`	the title of the card
`...`	the body content of the card
`footer`	the footer of the card; will be hidden if `footer=NULL`
`tools`	a list of tools or badges to be displayed at top-right corner, generated by `as_badge` or `card_tool`
`inputId`	the id of the card
`class`	the 'HTML' class of the entire card
`class_header`	the the 'HTML' class of the card header
`class_body`	the the 'HTML' class of the card body
`class_foot`	the the 'HTML' class of the card footer
`style_header`	'CSS' style of the header
`style_body`	'CSS' style of the body
`start_collapsed`	whether the card starts as collapsed
`resizable`	whether the card body can be resized vertically; notice that if true, then the default padding for body will be zero
`root_path`	see `template_root`
`body_main, body_side`	used by `card2`, the body content of the front and back sides of the card
`session`	shiny session domain
`method`	action to expand, minimize, or remove the cards; choices are `"collapse"`, `"expand"`, `"remove"`, `"toggle"`, `"maximize"`, `"minimize"`, and `"toggleMaximize"`

Value

'HTML' tags

Examples

```
library(shiny)
library(shidashi)

# Used for example only
```

```
ns <- I
session <- MockShinySession$new()

# -------------- Basic usage -------------
card(
  title = "Badges", div(
    class = "padding-20",
    p(
      "Add badges to the top-right corder. ",
      "Use \"|\" to indicate the badge classes; ",
      "for example: \"badge-info\", \"badge-warning\"..."
    ),
    hr(), p(
      "Use `resizable = TRUE` to make card resizable."
    )
  ),
  tools = list(
    as_badge("New|badge-info"),
    as_badge("3|badge-warning")
  ),
  class_body = "height-300",
  resizable = TRUE
)

# ---------------- With tools -----------------
card(
  title = "Default Tools",
  plotOutput(
    ns("card_defaulttool_plot"),
    height = "100%"
  ),
  tools = list(
    card_tool(
      widget = "link",
      href = "https://github.com/dipterix"
    ),
    card_tool(widget = "collapse"),
    card_tool(widget = "maximize")
  ),
  class_body = "height-300",
  resizable = TRUE
)

# -------------- Card2 example --------------
card2(
  title = "Card2 Example", body_main =
    plotOutput(
    outputId = ns("card2_plot"),
    height = "100%"
  ),
  body_side = fluidRow(
    column(
      6L, textInput(
```

```
        ns("card2_plot_title"),
        "Plot title"
      )
    ),
    column(
      6L, sliderInput(
        ns("card2_plot_npts"),
        "# of points", min = 1, max = 100,
        value = 10, step = 1, round = TRUE
      )
    )
  ),
  tools = list(
    card_tool(widget = "link",
              href = "https://github.com/dipterix"),
    card_tool(widget = "collapse"),
    card_tool(widget = "maximize")
  ),
  class_body = "height-300"
)
```

`card_tabset` *Generates a set of card panels*

Description

To insert, remove, or active card panels, see `card_tabset_operate`.

Usage

```
card_tabset(
  ...,
  inputId = rand_string(),
  title = NULL,
  names = NULL,
  active = NULL,
  tools = NULL,
  footer = NULL,
  class = "",
  class_header = "",
  class_body = "",
  class_foot = ""
)
```

Arguments

`...`	'HTML' tags; each tag will be placed into a card
`inputId`	the id of the card-set

`title`	the title of the card-set
`names`	title of the tabs
`active`	the title that will be active on load
`tools`	a list of tools or badges generated by `card_tool` or `as_badge`
`footer`	the footer element of the card-set
`class`	the 'HTML' class the of card-set
`class_header, class_body, class_foot`	additional 'HTML' class the of card header, body, and footer accordingly

Value

'HTML' tags

See Also

`card_tabset_operate`

Examples

```
library(shiny)
library(shidashi)

# Fake session to operate on card_tabset without shiny
session <- MockShinySession$new()

card_tabset(
  inputId = "card_set",
  title = "Cardset with Tools",
  `Tab 1` = p("Tab content 1"),
  class_body = "height-500",
  tools = list(
    as_badge(
      "New|badge-success"
    ),
    card_tool(
      widget = "collapse"
    ),
    card_tool(
      widget = "maximize"
    )
  )
)

card_tabset_insert(
  inputId = "card_set",
  title = "Tab 2",
  p("New content"),
  session = session
)
```

```
card_tabset_activate(
  inputId = "card_set",
  title = "Tab 1",
  session = session
)

card_tabset_remove(
  inputId = "card_set",
  title = "Tab 2",
  session = session
)
```

`card_tabset_operate` *Add, active, or remove a card within* `card_tabset`

Description

Add, active, or remove a card within `card_tabset`

Usage

```
card_tabset_insert(
  inputId,
  title,
  ...,
  active = TRUE,
  notify_on_failure = TRUE,
  session = shiny::getDefaultReactiveDomain()
)

card_tabset_remove(
  inputId,
  title,
  notify_on_failure = TRUE,
  session = shiny::getDefaultReactiveDomain()
)

card_tabset_activate(
  inputId,
  title,
  notify_on_failure = TRUE,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

inputId	the element id of card_tabset
title	the title of the card to insert, activate, or to remove
...	the content of the card
active	whether to set the card to be active once added
notify_on_failure	whether to show notifications on failure
session	shiny session domain

Value

These functions execute session$sendCustomMessage and return whatever value generated by that function; usually nothing.

See Also

card_tabset

---

card_tool *Generates small icon widgets*

Description

The icons cane be displayed at header line within accordion, card, card2, card_tabset. See their examples.

Usage

```
card_tool(
  inputId = NULL,
  title = NULL,
 widget = c("maximize", "collapse", "remove", "flip", "refresh", "link", "custom"),
  icon,
  class = "",
  href = "#",
  target = "_blank",
  start_collapsed = FALSE,
  ...
)
```

Arguments

inputId	the button id, only necessary when widget is "custom"
title	the tip message to show when the mouse cursor hovers on the icon
widget	the icon widget type; choices are "maximize", "collapse", "remove", "flip", "refresh", "link", and "custom"; see 'Details'
icon	icon to use if you are unsatisfied with the default ones
class	additional class for the tool icons
href, target	used when widget is "link", will open an external website; default is open a new tab
start_collapsed	used when widget is "collapse", whether the card should start collapsed
...	passed to the tag as attributes

Details

There are 7 widget types:

"maximize" allow the elements to maximize themselves to full-screen

"collapse" allow the elements to collapse

"remove" remove a card or card2

"flip" used together with flip_box, to allow card body to flip over

"refresh" refresh all shiny outputs

"link" open a hyper-link pointing to external websites

"custom" turn the icon into a actionButton. in this case, inputId must be specified.

Value

'HTML' tags to be included in tools parameter in accordion, card, card2, card_tabset

---

clipboardOutput *Generates outputs that can be written to clipboards with one click*

Description

Generates outputs that can be written to clipboards with one click

Usage

```
clipboardOutput(
  outputId = rand_string(),
  message = "Copy to clipboard",
  clip_text = "",
  class = NULL,
  as_card_tool = FALSE
)

renderClipboard(
  expr,
  env = parent.frame(),
  quoted = FALSE,
  outputArgs = list()
)
```

Arguments

outputId	the output id
message	tool tip to show when mouse hovers on the element
clip_text	the initial text to copy to clipboards
class	'HTML' class of the element
as_card_tool	whether to make the output as card_tool
expr	expression to evaluate; the results will replace clip_text
env	environment to evaluate expr
quoted	whether expr is quoted
outputArgs	used to replace default arguments of clipboardOutput

Value

'HTML' elements that can write to clip-board once users click on them.

Examples

```
clipboardOutput(clip_text = "Hey there")
```

---

flex_container *Generate 'HTML' tags with 'flex' layout*

Description

Generate 'HTML' tags with 'flex' layout

Usage

```
flex_container(
  ...,
  style = NULL,
  direction = c("row", "column"),
  wrap = c("wrap", "nowrap", "wrap-reverse"),
 justify = c("flex-start", "center", "flex-end", "space-around", "space-between"),
  align_box = c("stretch", "flex-start", "center", "flex-end", "baseline"),
  align_content = c("stretch", "flex-start", "flex-end", "space-between",
    "space-around", "center")
)

flex_item(
  ...,
  style = NULL,
  order = NULL,
  flex = "1",
  align = c("flex-start", "flex-end", "center")
)
```

Arguments

`...`	for `flex_container`, it's elements of `flex_item`; for `flex_item`, `...` are shiny 'HTML' tags
`style`	the additional 'CSS' style for containers or inner items
`direction, wrap, justify, align_box, align_content`	'CSS' styles for 'flex' containers
`order, align, flex`	CSS' styles for 'flex' items

Value

'HTML' tags

Examples

```
x <- flex_container(
  style = "position:absolute;height:100vh;top:0;left:0;width:100%",
  flex_item(style = 'background-color:black;'),
  flex_item(style = 'background-color:red;')
)
# You can view it via `htmltools::html_print(x)`
```

flip_box *An 'HTML' container that can flip*

Description

An 'HTML' container that can flip

Usage

```
flip_box(
  front,
  back,
  active_on = c("click", "click-front", "manual"),
  inputId = NULL,
  class = NULL
)

flip(inputId, session = shiny::getDefaultReactiveDomain())
```

Arguments

front	'HTML' elements to show in the front
back	'HTML' elements to show when the box is flipped
active_on	the condition when a box should be flipped; choices are 'click': flip when double-click on both sides; 'click-front': only flip when the front face is double-clicked; 'manual': manually flip in R code (see {flip(inputId)} function)
inputId	element 'HTML' id; must be specified if active_on is not 'click'
class	'HTML' class
session	shiny session; default is current active domain

Value

flip_box returns 'HTML' tags; flip should be called from shiny session, and returns nothing

Examples

```
# More examples are available in demo

library(shiny)
library(shidashi)

session <- MockShinySession$new()

flip_box(front = info_box("Side A"),
         back = info_box("Side B"),
```

```
          inputId = 'flip_box1')

flip('flip_box1', session = session)
```

`format_text_r` *Get re-formatted* R *expressions in characters*

Description

Get re-formatted R expressions in characters

Usage

```
format_text_r(
  expr,
  quoted = FALSE,
  reformat = TRUE,
  width.cutoff = 80L,
  indent = 2,
  wrap = TRUE,
  args.newline = TRUE,
  blank = FALSE,
  ...
)

html_highlight_code(
  expr,
  class = NULL,
  quoted = FALSE,
  reformat = TRUE,
  copy_on_click = TRUE,
  width.cutoff = 80L,
  indent = 2,
  wrap = TRUE,
  args.newline = TRUE,
  blank = FALSE,
  ...,
  hover = c("overflow-visible-on-hover", "overflow-auto")
)
```

Arguments

`expr`	R expressions
`quoted`	whether `expr` is quoted
`reformat`	whether to reformat

`width.cutoff, indent, wrap, args.newline, blank, ...`
: passed to `tidy_source`

`class`
: class of <pre> tag

`copy_on_click`
: whether to copy to clipboard if user clicks on the code; default is true

`hover`
: mouse hover behavior

Value

`format_text_r` returns characters, `html_highlight_code` returns the 'HTML' tags wrapping expressions in <pre> tag

See Also

`get_construct_string`

Examples

```
s <- format_text_r(print(local({a<-1;a+1})))
cat(s)

x <- info_box("Message", icon = "cogs")
s <- format_text_r(get_construct_string(x),
                   width.cutoff = 15L, quoted = TRUE)
cat(s)
```

---

get_construct_string *Get* R *expression used to generate the 'HTML' tags*

Description

This function only works on the elements generated by this package

Usage

```
get_construct_string(x)
```

Arguments

`x`
: 'HTML' tags

Value

Quoted R expressions that can generate the 'HTML' tags

See Also

format_text_r

Examples

```
x <- info_box("Message")
get_construct_string(x)
```

`guess_body_class` *Guess the 'AdminLTE' body class for modules, used internally*

Description

Guess the 'AdminLTE' body class for modules, used internally

Usage

```
guess_body_class(cls)
```

Arguments

`cls`	the class string of the <body> tag in `'index.html'`

Value

The proposed class for <body> tag

`include_view` *Template function to include 'snippets' in the view folder*

Description

Store the reusing 'HTML' segments in the `views` folder. This function should be used in the `'index.html'` template

Usage

```
include_view(file, ..., .env = parent.frame(), .root_path = template_root())
```

Arguments

`file`	files in the template `views` folder
`...`	ignored
`.env, .root_path`	internally used

Value

rendered 'HTML' segments

Examples

```
## Not run:
# in your 'index.html' file
<html>
<header>
{{ shidashi::include_view("header.html") }}
</header>
<body>

</body>
<!-- Before closing html tag -->
{{ shidashi::include_view("footer.html") }}
</html>

## End(Not run)
```

`info_box` *Generates 'HTML' info box*

Description

Generates 'HTML' info box

Usage

```
info_box(
  ...,
  icon = "envelope",
  class = "",
  class_icon = "bg-info",
  class_content = "",
  root_path = template_root()
)
```

Arguments

`...`	box content
`icon`	the box icon; default is `"envelope"`, can be hidden by specifying `NULL`
`class`	class of the box container
`class_icon`	class of the icon
`class_content`	class of the box body
`root_path`	see `template_root`

Value

'HTML' tags

Examples

```
library(shiny)
library(shidashi)

info_box("Message", icon = "cogs")

info_box(
  icon = "thumbs-up",
  span(class = "info-box-text", "Likes"),
  span(class = "info-box-number", "12,320"),
  class_icon = "bg-red"
)

info_box("No icons", icon = NULL)
```

---

javascript-tunnel *The 'JavaScript' tunnel*

Description

The 'JavaScript' tunnel

Usage

```
register_session_id(
  session = shiny::getDefaultReactiveDomain(),
  shared_id = NULL,
  shared_inputs = NA
)

register_session_events(session = shiny::getDefaultReactiveDomain())

get_theme(event_data, session = shiny::getDefaultReactiveDomain())

get_jsevent(
  event_data,
  type,
  default = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

session	shiny reactive domain
shared_id	the shared id of the session, usually automatically set
shared_inputs	the input names to share to/from other sessions
event_data	a reactive value list returned by register_session_events
type	event type; see 'Details'
default	default value if type is missing

Details

The register_session_id should be used in the module server function. It registers a shared_id and a private_id to the session. The sessions with the same shared_id can synchronize their inputs, specified by shared_inputs even on different browser tabs.

register_session_events will read the session events from 'JavaScript' and passively update these information. Any the event fired by shidashi.broadcastEvent in 'JavaScript' will be available as reactive value. get_jsevent provides a convenient way to read these events provided the right event types. get_theme is a special get_jsevent that with event type "theme.changed".

Function register_session_id and register_session_events should be called at the beginning of server functions. They can be called multiple times safely. Function get_jsevent and get_theme should be called in reactive contexts (such as observe, observeEvent).

Value

register_session_id returns a list of function to control "sharing" inputs with other shiny sessions with the same shared_id. register_session_events returns a reactive value list that reflects the session state. get_jsevent returns events fired by shidashi.broadcastEvent in 'JavaScript'. get_theme returns a list of theme, foreground, and background color.

Examples

```
# shiny server function

library(shiny)
server <- function(input, output, session){
  sync_tools <- register_session_id(session = session)
  event_data <- register_session_events(session = session)

  # if you want to enable syncing. They are suspended by default
  sync_tools$enable_broadcast()
  sync_tools$enable_sync()

  # get_theme should be called within reactive context
  output$plot <- renderPlot({
    theme <- get_theme(event_data)
    mar(bg = theme$background, fg = theme$foreground)
    plot(1:10)
  })
```

```
}
```

`module_info` *Obtain the module information*

Description

Obtain the module information

Usage

```
module_info(root_path = template_root(), settings_file = "modules.yaml")

load_module(
  root_path = template_root(),
  request = list(QUERY_STRING = "/"),
  env = parent.frame()
)
```

Arguments

`root_path`	the root path of the website project
`settings_file`	the settings file containing the module information
`request`	'HTTP' request string
`env`	environment to load module variables into

Details

The module files are stored in `modules/` folder in your project. The folder names are the module id. Within each folder, there should be one `"server.R"`, `R/`, and a `"module-ui.html"`.

The `R/` folder stores R code files that generate variables, which will be available to the other two files. These variables, along with some built-ins, will be used to render `"module-ui.html"`. The built-in functions are

ns shiny name-space function; should be used to generate the id for inputs and outputs. This strategy avoids conflict id effectively.

.module_id a variable of the module id

module_title a function that returns the module label

The `"server.R"` has access to all the code in `R/` as well. Therefore it is highly recommended that you write each 'UI' component side-by-side with their corresponding server functions and call these server functions in `"server.R"`.

Value

A data frame with the following columns that contain the module information:

`id` module id, folder name

`order` display order in side-bar

`group` group menu name if applicable, otherwise `NA`

`label` the readable label to be displayed on the side-bar

`icon` icon that will be displayed ahead of label, will be passed to `as_icon`

`badge` badge text that will be displayed following the module label, will be passed to `as_badge`

`url` the relative 'URL' address of the module.

Examples

```
library(shiny)
module_info()

# load master module
load_module()

# load specific module
module_data <- load_module(
  request = list(QUERY_STRING = "/?module=module_id"))
env <- module_data$environment

if(interactive()){

# get module title
env$module_title()

# generate module-specific shiny id
env$ns("input1")

# generate part of the UI
env$ui()

}
```

`notification` *The 'Bootstrap' notification*

Description

The 'Bootstrap' notification

Usage

```
show_notification(
  message,
  title = "Notification!",
  subtitle = "",
  type = c("default", "info", "warning", "success", "danger", "white", "dark"),
  close = TRUE,
  position = c("topRight", "topLeft", "bottomRight", "bottomLeft"),
  autohide = TRUE,
  fixed = TRUE,
  delay = 5000,
  icon = NULL,
  collapse = "",
  session = shiny::getDefaultReactiveDomain(),
  class = NULL,
  ...
)

clear_notifications(class = NULL, session = shiny::getDefaultReactiveDomain())
```

Arguments

`message`	notification body content, can be 'HTML' tags
`title, subtitle`	title and subtitle of the notification
`type`	type of the notification; can be `"default"`, `"info"`, `"warning"`, `"success"`, `"danger"`, `"white"`, `"dark"`
`close`	whether to allow users to close the notification
`position`	where the notification should be; choices are `"topRight"`, `"topLeft"`, `"bottomRight"`, `"bottomLeft"`
`autohide`	whether to automatically hide the notification
`fixed`	whether the position should be fixed
`delay`	integer in millisecond to hide the notification if `autohide=TRUE`
`icon`	the icon of the title
`collapse`	if `message` is a character vector, the collapse string
`session`	shiny session domain
`class`	the extra class of the notification, can be used for style purposes, or by `clear_notifications` to close specific notification types.
`...`	other options; see https://adminlte.io/docs/3.1//javascript/toasts.html#options

Value

Both functions should be used in shiny reactive contexts. The messages will be sent to shiny 'JavaScript' interface and nothing will be returned.

Examples

```
## Not run:

# the examples must run in shiny reactive context

show_notification(
  message = "This validation process has finished. You are welcome to proceed.",
  autohide = FALSE,
  title = "Success!",
  subtitle = "type='success'",
  type = "success"
)

show_notification(
  message = "This notification has title and subtitle",
  autohide = FALSE,
  title = "Hi there!",
  subtitle = "Welcome!",
  icon = "kiwi-bird",
  class = "notification-auto"
)

# only clear notifications with class "notification-auto"
clear_notifications("notification-auto")


## End(Not run)
```

`progressOutput` *Progress bar in shiny dashboard*

Description

For detailed usage, see demo application by running `render()`.

Usage

```
progressOutput(
  outputId,
  ...,
  description = "Initializing",
  width = "100%",
  class = "bg-primary",
  value = 0,
  size = c("md", "sm", "xs")
)

renderProgress(expr, env = parent.frame(), quoted = FALSE, outputArgs = list())
```

Arguments

outputId	the element id of the progress
...	extra elements on the top of the progress bar
description	descriptive message below the progress bar
width	width of the progress
class	progress class, default is "bg-primary"
value	initial value, ranging from 0 to 100; default is 0
size	size of the progress bar; choices are "md", "sm", "xs"
expr	R expression that should return a named list of value and description
env	where to evaluate expr
quoted	whether expr is quoted
outputArgs	a list of other parameters in progressOutput

Value

progressOutput returns 'HTML' tags containing progress bars that can be rendered later via shiny_progress or renderProgress. renderProgress returns shiny render functions internally.

Examples

```
library(shiny)
library(shidashi)
progressOutput("sales_report_prog1",
               description = "6 days left!",
               "Add Products to Cart",
               span(class="float-right", "123/150"),
               value = 123/150 * 100)

# server function
server <- function(input, output, session, ...){
  output$sales_report_prog1 <- renderProgress({
    return(list(
      value = 140 / 150 * 100,
      description = "5 days left!"
    ))
  })
}
```

render *Render a 'shidashi' project*

Description

Render a 'shidashi' project

Usage

```
render(
  root_path = template_root(),
  ...,
  launch_browser = TRUE,
  as_job = TRUE,
  test_mode = getOption("shiny.testmode", FALSE)
)
```

Arguments

root_path	the project path, default is the demo folder from `template_root()`
...	additional parameters passed to runApp, such as `host`, `port`
launch_browser	whether to launch browser; default is `TRUE`
as_job	whether to run as 'RStudio' jobs; this options is only available when 'RStudio' is available
test_mode	whether to test the project; this options is helpful when you want to debug the project without relaunching shiny applications

Value

This functions runs a 'shiny' application, and returns the job id if 'RStudio' is available.

Examples

```
template_root()

if(interactive()){
  render()
}
```

shiny_progress *Wrapper of shiny progress that can run without shiny*

Description

Wrapper of shiny progress that can run without shiny

Usage

```
shiny_progress(
  title,
  max = 1,
  ...,
  quiet = FALSE,
  session = shiny::getDefaultReactiveDomain(),
  shiny_auto_close = FALSE,
  log = NULL,
  outputId = NULL
)
```

Arguments

title	the title of the progress
max	max steps of the procedure
...	passed to initialization method of Progress
quiet	whether the progress needs to be quiet
session	shiny session domain
shiny_auto_close	whether to close the progress once function exits
log	alternative log function
outputId	the element id of progressOutput, or NULL to use the default shiny progress

Value

a list of functions that controls the progress

Examples

```
{
  progress <- shiny_progress("Procedure A", max = 10)
  for(i in 1:10){
    progress$inc(sprintf("Step %s", i))
    Sys.sleep(0.1)
  }
  progress$close()
```

```
}

if(interactive()){
  library(shiny)

  ui <- fluidPage(
    fluidRow(
      column(12, actionButton("click", "Click me"))
    )
  )

  server <- function(input, output, session) {
    observeEvent(input$click, {
      progress <- shiny_progress("Procedure B", max = 10,
                                 shiny_auto_close = TRUE)
      for(i in 1:10){
        progress$inc(sprintf("Step %s", i))
        Sys.sleep(0.1)
      }
    })
  }

  shinyApp(ui, server)
}
```

---

show_ui_code *Used by demo project to show the generating code*

Description

Please write your own version. This function is designed for demo-use only.

Usage

```
show_ui_code(
  x,
  class = NULL,
  code_only = FALSE,
  as_card = FALSE,
  card_title = "",
  class_body = "bg-gray-70",
  width.cutoff = 80L,
  indent = 2,
  wrap = TRUE,
  args.newline = TRUE,
  blank = FALSE,
  copy_on_click = TRUE,
```

```
  ...
)
```

Arguments

x	'HTML' tags generated by this package
class	additional 'HTML' class
code_only	whether to show code only
as_card	whether to wrap results in card
card_title, class_body	used by card if as_card=TRUE
width.cutoff, indent, wrap, args.newline, blank, copy_on_click, ...	passed to html_highlight_code

Value

'HTML' tags

See Also

html_highlight_code

---

template_settings *Configure template options that are shared across the sessions*

Description

Configure template options that are shared across the sessions

Usage

```
template_settings

template_settings_set(...)

template_settings_get(name, default = NULL)

template_root()
```

Arguments

...	key-value pair to set options
name	character, key of the value
default	default value if the key is missing

Format

An object of class `list` of length 3.

Details

The settings is designed to store static key-value pairs that are shared across the sessions. The most important key is `"root_path"`, which should be a path pointing to the template folder.

Value

`template_settings_get` returns the values represented by the corresponding keys, or the default value if key is missing.

Examples

```
# Get current website root path

template_root()
```

---

use_template *Download 'shidashi' templates from 'Github'*

Description

Download 'shidashi' templates from 'Github'

Usage

```
use_template(path, user = "dipterix", theme = "AdminLTE3", ...)
```

Arguments

`path`	the path to create 'shidashi' project
`user`	'Github' user name
`theme`	the theme to download
`...`	ignored

Details

To publish a 'shidashi' template, create a 'Github' repository called `'shidashi-templates'`, or fork the built-in templates. The `theme` is the sub-folder of the template repository.

An easy way to use a template in your project is through the 'RStudio' project widget. In the 'RStudio' navigation bar, go to "File" menu, click on the "New Project..." button, select the "Create a new project" option, and find the item that creates 'shidashi' templates. Use the widget to set up template directory.

Value

the target project path

Index