

Package ‘spsComps’

May 4, 2021

Title 'systemPipeShiny' UI and Server Components

Version 0.2

Date 2021-05-03

Description The systemPipeShiny (SPS) framework comes with many UI and server components. However, installing the whole framework is heavy and takes some time. If you would like to use UI and server components from SPS in your own Shiny apps, but do not want to install the whole framework, just install this package.

Depends R (>= 4.0.0), shiny (>= 1.5.0), spsUtil

Imports assertthat, stringr, glue (>= 1.4.0), magrittr, bsplus, shinyFiles, shinyWidgets, shinytoastr, shinyAce, htmltools, vroom, dplyr, utils, R6, crayon

Suggests testthat, shinyjqui

License GPL (>= 3)

Encoding UTF-8

BugReports <https://github.com/lz100/spsComps/issues>

URL <https://github.com/lz100/spsComps>

RoxygenNote 7.1.1

Config/testthat/edition 3

NeedsCompilation no

Author Le Zhang [aut, cre]

Maintainer Le Zhang <lzhang100@gmail.com>

Repository CRAN

Date/Publication 2021-05-04 16:50:05 UTC

R topics documented:

addLoader	2
animateIcon	7

animateUI	9
bsHoverPopover	12
clearableTextInput	13
cssLoader	14
dynamicFile	16
gallery	19
heightMatcher	21
hexLogo	22
hrefTab	25
hrefTable	26
loadDF	29
pgPaneUI	31
renderDesc	32
shinyCatch	33
shinyCheckPkg	36
spsCodeBtn	37
spsDepend	40
spsGoTop	42
spsHr	43
spsTimeline	44
spsValidate	45
tabTitle	48
textButton	48
textInputGroup	50

Index	52
--------------	-----------

addLoader	<i>Add CSS loaders from server</i>
-----------	------------------------------------

Description

Add/remove CSS loaders from server to any Shiny/HTML component. It is useful to indicate busy status when some code is running in the server and when it finishes, remove the loader to indicate clear status.

Value

CSS load in R6 class

Methods

Public methods:

- [addLoader\\$new\(\)](#)
- [addLoader\\$show\(\)](#)
- [addLoader\\$hide\(\)](#)
- [addLoader\\$clone\(\)](#)

Method new(): create a loader object

Usage:

```
addLoader$new(
  target_selector = "",
  isID = TRUE,
  type = "default",
  id = "",
  height = NULL,
  width = height,
  color = "#337ab7",
  opacity = 1,
  method = "replace",
  block = TRUE,
  center = TRUE,
  bg_color = "#eee",
  footer = NULL,
  z_index = 2000,
  alert = FALSE,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments:

`target_selector` string, which Shiny component you want to add the loader to? a shiny component ID or a valid CSS selector if `isID = FALSE`. for example, you have a button and want to add animation to it:

```
actionButton(inputId = "btn")
```

This function is used in server only, so if you are in shiny module, **DO NOT** add the `ns()` wrapper.

`isID` bool, is your selector an ID?

`type` string, one of "circle", "dual-ring", "facebook", "heart", "ring", "roller", "default", "ellipsis", "grid", "hourglass", "ripple", "spinner", default is "default".

`id` string, the unique ID for the loader, if not provided, a random ID will be given. If you are using shiny modules, use `session$ns('YOUR_ID')` to wrap it.

`height` string, (r)em, "1.5rem", "1.5em", or pixel, like "10px". Default is NULL, will be automatically calculated based on the target component. It is recommend to use NULL for "replace" and "inline" method to let it automatically be calculated, but required for "full_screen" method.

`width` string, default is the same as height to make it square.

`color` string, any valid CSS color name, or hex color code

`opacity` number, between 0-1

`method` one of "replace", "inline", "full_screen", see details

`block` bool, for some input components, once the loader starts, it can also block user interaction with the component, very useful for "inline" method, eg. prevent users from clicking the button while some process is still running.

`center` bool, try to place the load to the center of the target for "inline" and "replace" and center of the screen for "full_screen".

`bg_color` string, any valid CSS color name, or hex color code. Only works for "full_screen" method.

`footer` Additional Shiny/HTML component to add below the loader, like a title `h1("load title")`. `inline` method does not have a footer.

`z_index` number, only works for "full_screen" method, what CSS layer should the overlay be places. In HTML, all elements have the default of 0.

`alert` bool, should alert if target cannot be found or other javascript errors? mainly for debugging

`session` shiny session

Details:

Methods:

- `replace`: use a HTML div with the same CSS styles to **replace the original target**, but add the loader inside and remove original content inside. When the loader is hide, show the original div and hide this loader div. Height and width is the original div's height unless specially specified. Good example of this will be some plot outputs.
- `inline`: append the loader as the first child of target HTML container. loader's height and width is the original div's height unless specially specified. In addition, this methods will **disable** all inputs and buttons inside the target container, so this method can be useful on some buttons.
- `full_screen`: Do not change anything of the target HTML container, add an overlay to **cover the whole page** when show and hide the overlay when hide. This method requires the height to be specified manually. Under this method, `bg_color` and `z_index` can also be changed.

Returns: A R6 loader object

Method `show()`: show the loader

Usage:

```
addLoader$show(alert = FALSE)
```

Arguments:

`alert` bool, if the target selector or loader is not found, alert on UI? For debugging purposes.

Method `hide()`: hide the loader

Usage:

```
addLoader$hide(alert = FALSE)
```

Arguments:

`alert` bool, if the target selector or loader is not found, alert on UI? For debugging purposes.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
addLoader$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.


```

    loader_inline$show()
  })
  observeEvent(input$b_in_stop, {
    loader_inline$hide()
  })
  ## full screen
  observeEvent(input$b_fs_start, {
    loader_fs$show()
    Sys.sleep(2)
    loader_fs$hide()
  })
  ## chunk
  observeEvent(input$chunk_start, {
    loader_chunk$show()
  })
  observeEvent(input$chunk_stop, {
    loader_chunk$hide()
  })
}

shinyApp(ui, server)
}

if (interactive()){
  ui <- bootstrapPage(
    spsDepend("css-loader"),
    h4("Add loaders to Shiny `render` events"),
    tags$b("Replace"), br(),
    selectInput(inputId = "n_re",
      label = "Change this to render the following plot",
      choices = c(10, 20, 35, 50)),
    plotOutput(outputId = "p_re"),
    br(), tags$b("Full screen"), br(),
    selectInput(inputId = "n_fs",
      label = "Change this to render the following plot",
      choices = c(10, 20, 35, 50)),
    plotOutput(outputId = "p_fs")
  )

  server <- function(input, output, session) {
    # create loaders
    p_re <- addLoader$new("p_re", type = "facebook")
    p_fs <- addLoader$new(
      "p_fs", color = "pink", method = "full_screen",
      bg_color = "#eee", height = "30rem", type = "grid",
      footer = h4("Replotting...")
    )
    # use loaders in rendering
    output$p_re <- renderPlot({
      on.exit(p_re$hide())
      p_re$show()
      Sys.sleep(1)
    })
  }
}

```

```

    hist(faithful$eruptions,
         probability = TRUE,
         breaks = as.numeric(input$n_re),
         xlab = "Duration (minutes)",
         main = "Geyser eruption duration")
  })
  output$p_fs <- renderPlot({
    on.exit(p_fs$hide())
    p_fs$show()
    Sys.sleep(1)
    hist(faithful$eruptions,
         probability = TRUE,
         breaks = as.numeric(input$n_fs),
         xlab = "Duration (minutes)",
         main = "Geyser eruption duration")
  })
}
shinyApp(ui, server)
}

```

animateIcon

Font awesome animated icons

Description

Greatly enhance the `shiny::icon` with animations. Built on top of [font-awesome-animation](#).

Usage

```

animateIcon(
  name,
  animation = NULL,
  speed = NULL,
  hover = FALSE,
  color = "",
  size = NULL
)

```

Arguments

name	string, the name of the font-awesome icon
animation	what kind of animation you want, one of "wrench", "ring", "horizontal", "horizontal-reverse", "vertical", "flash", "bounce", "bounce-reverse", "spin", "spin-reverse", "float", "pulse", "shake", "tada", "passing", "passing-reverse", "burst", "falling", "falling-reverse", "rising". See https://l-lin.github.io/font-awesome-animation/#animation-list or our online demo for details.
speed	string, one of "fast", "slow"
hover	bool, trigger animation on hover?

color string, color of the icon, a valid color name or hex code
size string, change font-awesome icon size, one of "xs", "sm", "lg", "2x", "3x", "5x", "7x", "10x". See examples.

Details

If you don't specify any animation, it will work the same as the original `shiny::icon` function. Fully compatible with any shiny functions that requires an icon as input.

Value

a icon tag

Examples

```

if(interactive()){
  library(shiny)

  ui <- fluidPage(
    style = "text-align: center;",
    tags$label("same as original icon function"), br(),
    animateIcon("home"), br(),
    tags$label("Change animation and color"), br(),
    animateIcon(
      name = "home", animation = "horizontal", speed = "slow", color = "red"
    ), br(),
    tags$label("work in a button"), br(),
    actionButton(
      "a", "a", icon = animateIcon("spinner", "spin", "fast")
    ), br(),
    tags$label("hover your mouse on the next one"), br(),
    animateIcon(
      name = "wrench", animation = "wrench", hover = TRUE, color = "green"
    ), br(),
    tags$label("change size"), br(),
    animateIcon("home"),
    animateIcon("home", size = "xs"),
    animateIcon("home", size = "sm"),
    animateIcon("home", size = "lg"),
    animateIcon("home", size = "2x"),
    animateIcon("home", size = "3x"),
    animateIcon("home", size = "5x"),
    animateIcon("home", size = "7x"),
    animateIcon("home", size = "10x")
  )

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

  }

  shinyApp(ui, server)
}

```

animateUI	<i>Add/remove animation to any HTML/shiny component</i>
-----------	---

Description

Add animation to a HTML or component and remove it

Usage

```
animateUI(selector, animation = NULL, speed = NULL, hover = FALSE, isID = TRUE)
```

```
animateServer(
  selector,
  animation = NULL,
  speed = NULL,
  hover = FALSE,
  isID = TRUE,
  session = shiny::getDefaultReactiveDomain()
)
```

```
animationRemove(
  selector,
  isID = TRUE,
  alert = FALSE,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

selector	<p>string, a shiny component ID or a valid CSS selector if isID = FALSE. for example, you have a button and want to add animation to it:</p> <pre>actionButton(inputId = "btn")</pre> <p>Then the selector is "btn" selector = 'btn'. If you are using shiny modules, use ns() to wrap it in UI for the button <code>actionButton(inputId = ns("btn"))</code>, and also add ns() to selector <code>selector = ns('btn')</code> for the <code>animateUI</code> function. If you are using the server side functions <code>animateServer</code> and <code>animationRemove</code>, DO NOT add the ns() wrapper.</p>
animation	<p>what kind of animation you want, one of "wrench", "ring", "horizontal", "horizontal-reverse", "vertical", "flash", "bounce", "bounce-reverse", "spin", "spin-reverse", "float", "pulse", "shake", "tada", "passing", "passing-reverse", "burst", "falling", "falling-reverse", "rising". See https://l-lin.github.io/font-awesome-animation/#animation-list or our online demo for details.</p>
speed	string, one of "fast", "slow"
hover	bool, trigger animation on hover?
isID	bool, is your selector an ID?

session	the current shiny session
alert	bool, for <code>animationRemove</code> only: if the component is not found or it does not contain any animation or the animation is not added by <code>spsComps</code> , alert on UI? More like for debugging purposes.

Details

- `animateUI`: use on the UI side, which means add the animation when UI loads complete.
- `animateServer`: use on the server side. Use server to trigger the animation on a component at some point.
- `animationRemove`: use on the server side, to remove animation on a certain component.

Selector:

Usually for beginners use the shiny component ID is enough, but sometimes a HTML element may not has the 'id' attribute. In this case, you can still animate the element by advanced CSS selector. For these selectors, turn off the `isID = FALSE` and provide the selector in a single string. Google "CSS selector" to learn more.

only server functions:

If you use `animateServer` or `animationRemove` on the server, you need to add `spsDepend("animation")` somewhere in your UI to load the required CSS and javascript, inside `tags$head()` is preferred. see examples.

Value

see details

Examples

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

  ui <- fluidPage(
    spsDepend("animation"),
    column(
      6,
      h3("Adding animations from UI"),
      tags$label("to a button"), br(),
      actionButton("btn1", "random button"), br(),
      animateUI("btn1", animation = "ring"),
      tags$label("to some text"), br(),
      p(id = "mytext", class = "text-red", "some move text"), br(),
      animateUI("mytext", animation = "horizontal", speed = "fast"),
      tags$label("on hover, move mouse on the red thumb"), br(),
      actionButton(
        "btn2", "",
        icon = icon(id = "myicon", "thumbs-o-up"),
        style = "color: red; boarder: initial; border-color: transparent;"
      ), br(),
      animateUI("btn2", animation = "bounce", speed = "fast", hover = TRUE),
```

```

    tags$label("on a plot"), br(),
    plotOutput("plot1"),
    animateUI("plot1", animation = "float", speed = "fast")
  ),
  column(
    6,
    h3("Adding/removing animations from server"),
    tags$label("use a button to control"), br(),
    actionButton("btn3", "animate itself"),
    actionButton("btn4", "stop animation"), br(),
    tags$label("advanced selector in for complex group"), br(),
    sliderInput(
      "myslider",
      label = "animating if less than 5",
      value = 0,
      min = 0, max = 10,
      step = 1
    ),
    sliderInput(
      "myslider2", min = 0, max = 10, value = 10,
      label = "this one will not be selected"
    )
  )
)
)

server <- function(input, output, session) {
  output$plot1 <- renderPlot(plot(1:10, 10:1))
  observeEvent(input$myslider, {
    if (input$myslider <= 5) {
      animateServer(
        # the slider container does not has the ID, it is inside
        selector = ".shiny-input-container:has(#myslider)",
        animation = "horizontal", speed = "slow", isID = FALSE
      )
    } else {
      animationRemove(
        selector = ".shiny-input-container:has(#myslider)",
        isID = FALSE
      )
    }
  })
  observeEvent(input$btn3, {
    animateServer("btn3", animation = "flash", speed = "slow")
  })
  observeEvent(input$btn4, {
    animationRemove("btn3")
  })
}

shinyApp(ui, server)
}

```

bsHoverPopover *Bootstrap popover trigger on hover instead of click*

Description

enhanced Bootstrap 3 popover by hovering, see [bsplus::bs_embed_popover](#) for details. Everything is similar but has additional trigger method, default "hover". Original method only works with "click" on buttons.

Usage

```
bsHoverPopover(
  tag,
  title = NULL,
  content = NULL,
  placement = "top",
  trigger = "hover",
  ...
)
```

Arguments

tag	htmltools::tag , generally htmltools::tags\$button() or htmltools::tags\$a() , or shiny::actionButton()
title	character, title for the popover, generally text
content	character, content for the popover body, can be HTML
placement	character, placement of the popover with respect to tag
trigger	trigger method, default "hover", one of click hover focus manual.
...	other named arguments, passed to bsplus::bs_set_data()

Value

shiny element

Examples

```
if(interactive()){
  library(shiny)
  library(magrittr)
  ui <- fluidPage(
    column(2),
    column(
      8,
      actionButton('a', 'On button') %>%
        bsHoverPopover(
          title = "title a",
          content = "popover works on a button",
          placement = "bottom"
        )
    )
  )
}
```

```

    ),
    tags$a("On link") %>%
      bsHoverPopover(
        title = "title b",
        content = "popover works on a link",
        placement = "bottom"
      ),
    div(
      tags$b("general element"),
      style =
        ,
        height: 100px;
        background-color: cornflowerblue;
        ,
    ) %>%
      bsHoverPopover(
        title = "general element",
        content = "popover works on a 'div'",
        placement = "right"
      )
    )
  )
  server <- function(input, output, session) {}
  shinyApp(ui, server)
}

```

clearableTextInput *A clearable text input control*

Description

An UI component with a "X" button in the end to clear the entire entered text. It works the same as Textinput.

Usage

```

clearableTextInput(
  inputId,
  label = "",
  value = "",
  placeholder = "",
  style = "width: 100%;"
)

```

Arguments

inputId	ID
label	text label above

value	default value
placeholder	place holder text when value is empty
style	additional CSS styles you want to apply

Value

a shiny component

Examples

```

if(interactive()){
  ui <- fluidPage(
    clearableTextInput("input1", "This is a input box", style = "width: 50%;"),
    verbatimTextOutput("out1")
  )
  server <- function(input, output, session) {
    output$out1 <- renderPrint(input$input1)
  }
  shinyApp(ui, server)
}

```

cssLoader

Create a variety of CSS loaders on UI

Description

CSS loaders can improve user experience by adding a small animation icon to a HTML element. `spsComps` provides you 12 different looking CSS loaders. Unlike other Shiny packages, you have full control of the CSS loader here, like position, color, size, opacity, etc.

Usage

```

cssLoader(
  type = "default",
  id = "",
  height = "1.5rem",
  width = height,
  color = "#337ab7",
  opacity = 1,
  inline = FALSE,
  is_icon = FALSE,
  ...
)

```

Arguments

type	string, one of "circle", "dual-ring", "facebook", "heart", "ring", "roller", "default", "ellipsis", "grid", "hourglass", "ripple", "spinner"
id	string, optional, ID for the component, if not given, a random ID will be given.
height	string, pixel, like "10px"; or (r)em, "1.5rem", "1.5em". Default is "1.5rem".
width	string, default is the same as height. For most loader, you want to keep width = height for a square shape.
color	string, any valid CSS color name, or hex color code
opacity	number, between 0-1
inline	bool, do you want the loader be inline? This is useful to turn on if you want to add the loader to a <code>shiny::actionButton</code> , so the loader and button label will be on the same line. See examples.
is_icon	bool, default uses the HTML div tag, turn on this option will use the <code>i</code> tag for icon. Useful if you want to add the loader as icon argument for the <code>shiny::actionButton</code> . See examples.
...	other shiny tags or HTML attributes you want to add to the loader.

Details**'rem' unit:**

For most modern web apps, including Shiny, 1rem = 10px

Value

returns a css loader component.

Examples

```
if (interactive()){
  library(shiny)
  heights <- paste0(c(1.5, 3, 5, 8, 10, 15, 20), "rem")
  colors <- list(
    colorRampPalette(c("#00d2ff", "#3a7bd5"))(7),
    colorRampPalette(c("#59C173", "#a17fe0", "#5D26C1"))(7),
    colorRampPalette(c("#667db6", "#0082c8", "#5D26C1", "#667db6"))(7),
    colorRampPalette(c("#f2709c", "#ff9472"))(7),
    colorRampPalette(c("#FC5C7D", "#6A82FB"))(7),
    colorRampPalette(c("#4568DC", "#B06AB3"))(7)
  )
  types <- c("circle", "dual-ring", "facebook", "heart",
            "ring", "roller", "default", "ellipsis",
            "grid", "hourglass", "ripple", "spinner")
  ui <- fluidPage(
    lapply(seq_along(types), function(i){
      div(
        h4(types[i]), br(),
        lapply(1:7, function(x){
          cssLoader(
```

```

        types[i], height = heights[x],
        color = colors[[if(i > 6) i - 6 else i]][x],
        inline = TRUE
      )
    }},
  br()
)
})
)
server <- function(input, output, session) {}
shinyApp(ui, server)
}

# use with buttons
if (interactive()){
  library(shiny)
  ui <- fluidPage(
    actionButton(
      "btn-a", "",
      ## `inline = TRUE` is important if you want loader and
      ## text in the same line.
      icon = cssLoader(is_icon = TRUE, inline = TRUE, color = "#3a7bd5"
    )
  ),
  actionButton(
    "btn-b", "Loading",
    icon = cssLoader(is_icon = TRUE, color = "#667db6", inline = TRUE)
  )
)
server <- function(input, output, session) {}
shinyApp(ui, server)
}

```

dynamicFile

Dynamically generate Shiny file selection component based on option

Description

Depending on the "mode" in SPS options, this function renders a similar UI components but behaves differently on server.

1. local mode will not copy file, directly use a path pointer.
2. server mode upload file and store in temp. Expect similar behavior as [shiny::fileInput](#).

Usage

```

dynamicFile(
  id,
  title = "Select your file:",
  label = "Browse",

```



```

    icon = NULL,
    style = "",
    multiple = FALSE
  )

  dynamicFileServer(input, session, id, roots = c(root = "default"))

```

Arguments

id	element ID, Use <code>ns()</code> to wrap the id if you are using within a shiny module, but DO NOT use <code>ns()</code> to wrap the id on server side
title	element title
label	upload button label
icon	button icon, only works for local mode
style	additional button style, only works for local mode
multiple	bool, are multiple files allowed?
input	shiny server input
session	shiny server session
roots	a named character vector, paths where users can reach on the server, so only required for "server" mode, default is current directory + all system volumes. You can lock users to a specific path, so they are not allowed to browse parent folders. like only current directory: <code>c(current=getwd())</code> ; a temp folder: <code>c(current=tempdir())</code> ; unlimited: <code>c(shinyFiles::getVolumes())</code>

Details

To setup the option:

The local mode uses functions from [shinyFiles](#) so it will reach file system on the **server end**. Although the latest [shinyFiles](#) limits users to only specified server end location (folder), there is still some **risk**. That's why it is named "local", you are encouraged to run the app on your local computer. The advantage of "local" is: for some very large files, it does not upload and store in the temp. Rather, it directly parses the path on the local file system and return the path immediately. It means the file has to exist on the file system that serves the Shiny app. If you deploy the app on places like shinyapps.io, users can only choose files from server.

On the other hand, server mode uses original shiny default upload component. Users can upload files from local to server. So users do not have access to server end file system if you deploy it online. However, the limitations are:

1. not ideal for large files, default limit is 30MB, and there is no break point upload.
2. If you are running the app on your own computer, local end and server end is the same, which is your computer. Using server mode will make a copy of your existing file to temp location which is a waste of time and storage.

To set up options:

1. Under SPS framework, edit options in `global.R`.

2. Outside SPS framework with your own Shiny app, use `spsUtil::spsOption()` function, like `spsUtil::spsOption("mode", "server")` or `spsUtil::spsOption("mode", "local")` to set up mode.

If you are not sure what mode you are on, use `spsUtil::spsOption('mode')` to check.

Value

a Shiny upload component on UI

For the server end it returns a **reactive** object which is a dataframe, need to extract the value inside reactive expression, observe, or inside isolate. See examples

Examples

```
# Simple example
if(interactive()){
  options(sps = list(mode='server')) # Change the mode to 'local' to see difference
  ui <- fluidPage(
    dynamicFile("server_file", "server"),
    verbatimTextOutput("server_out")
  )

  server <- function(input,output,session){
    file_server <- dynamicFileServer(input,session, id = "server_file")
    output$server_out <- renderPrint({
      file_server() # remember to use `()` for reactive value
    })
  }
  shinyApp(ui = ui, server = server)
}

# To demonstrate different modes in the same app, we can set options before the function.
# This is NOT recommended, you should stick with only one mode for the entire app.
if(interactive()){
  spsOption("mode", "local")
  local_ui <- dynamicFile("local_file", "local")
  spsOption("mode", "server")
  server_ui <- dynamicFile("server_file", "server")
  ui <- fluidPage(
    column(
      6,
      local_ui,
      verbatimTextOutput("local_out")
    ),
    column(
      6,
      server_ui,
      verbatimTextOutput("server_out")
    )
  )

  server <- function(input,output,session){
    spsOption("mode", "local")
```

```

file_local <- dynamicFileServer(input,session, id = "local_file")
output$local_out <- renderPrint({
  file_local() # remember to use `()` for reactive value
})
spsOption("mode", "server")
file_server <- dynamicFileServer(input,session, id = "server_file")
output$server_out <- renderPrint({
  file_server()
})
}
shinyApp(ui = ui, server = server)
}

```

 gallery

A shiny gallery component

Description

Create a gallery to display images or photos

texts, hrefs, images Must have the same length

If there is any image that you do not want to add links, use "" to occupy the space, e.g

hrefs = c("https://xxx.com", "", "https://xxx.com")

If the link is empty, there will be no hover effect on that image, and you cannot click it.

Similar to hrefs, for the texts, use "" to occupy space

Usage

```

gallery(
  texts,
  hrefs,
  images,
  Id = NULL,
  title = "Gallery",
  title_color = "#0275d8",
  image_frame_size = 4,
  enlarge = FALSE,
  enlarge_method = c("inline", "modal"),
  target_blank = FALSE,
  style = ""
)

```

Arguments

texts	vector of labels under each image
hrefs	vector of links when each image is clicked
images	a vector of image sources, can be online URLs or local resource paths.

Id	ID of this gallery
title	Title of gallery
title_color	Title color
image_frame_size	integer, 1-12, this controls width. How large is each image. 12 is the whole width of the screen and 1 is 1/12 of the screen. Consider numbers than can fully divide 12, like 1, 2, 3, 4, 6 or 12 (if you want only 1 image per row).
enlarge	bool, when click on the image, enlarge it? If enlarge is enabled, click the photo will enlarge instead of jump to the link. Only the title below contains the link if enlarge is enabled.
enlarge_method	how the photo is enlarged on click, one of "inline" – within the gallery change the size of photo to 12, "modal" – display photo in a pop-up modal.
target_blank	bool, whether to add target="_blank" to the link?
style	additional CSS style you want to add to the most outside component "div"

Details

modal **enlarge**:

When view the modal enlarged images, click the "X" button or anywhere outside the image to close the full screen view.

Value

a gallery component

Examples

```
if(interactive()){
  texts <- c("p1", "p2", "", "p4", "p5")
  hrefs <- c("https://github.com/lz100/spsComps/blob/master/img/1.jpg?raw=true",
            "https://github.com/lz100/spsComps/blob/master/img/2.jpg?raw=true",
            "",
            "https://github.com/lz100/spsComps/blob/master/img/4.jpg?raw=true",
            "https://github.com/lz100/spsComps/blob/master/img/5.jpg?raw=true")
  images <- c("https://github.com/lz100/spsComps/blob/master/img/1.jpg?raw=true",
             "https://github.com/lz100/spsComps/blob/master/img/2.jpg?raw=true",
             "https://github.com/lz100/spsComps/blob/master/img/3.jpg?raw=true",
             "https://github.com/lz100/spsComps/blob/master/img/4.jpg?raw=true",
             "https://github.com/lz100/spsComps/blob/master/img/5.jpg?raw=true")
  library(shiny)

  ui <- fluidPage(
    column(
      6,
      gallery(texts = texts, hrefs = hrefs, images = images, title = "Default gallery"),
      spsHr(),
      gallery(texts = texts, hrefs = hrefs, images = images,
              image_frame_size = 2, title = "Photo size"),
      spsHr(),
```

```

    gallery(texts = texts, hrefs = hrefs, images = images,
            enlarge = TRUE, title = "Inline enlarge"),
    spsHr(),
    gallery(
      texts = texts, hrefs = hrefs, images = images,
      enlarge = TRUE, title = "Modal enlarge",
      enlarge_method = "modal"
    )
  )
)
)

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

}

shinyApp(ui, server)
}

```

heightMatcher

Match height of one element to the other element

Description

Match the height of one element to the second element. If the height of second element change, the height of first element will change automatically

Usage

```
heightMatcher(div1, div2, isID = TRUE)
```

Arguments

div1	element ID, or jquery selector if isID = FALSE. The first element that you want to match the height to the other element
div2	matched element ID or selector, the other element
isID	bool, if TRUE, div1 and div2 will be treated as ID, otherwise you can use complex jquery selector

Value

will be run as javascript

Examples

```

if(interactive()){
  library(shiny)
  library(shinyjquery)
  ui <- fluidPage(
    column(

```

```

      3, id = "a",
      style = "border: 1px black solid; background-color: gray;",
      p("This block's height is matched with orange one")
    ),
    shinyjquery::jquery_resizable(column(
      2, id = "b",
      style = "border: 1px black solid; background-color: orange;",
      p("drag the bottom-right corner")
    )),
    column(
      3, id = "c",
      style = "border: 1px black solid; background-color: red;",
      p("This block's is not matched with others")
    ),
    heightMatcher("a", "b")
  )

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

}
# Try to drag `b` from bottom right corner and see what happens to `a`
shinyApp(ui, server)
}

```

 hexLogo

Hexagon logo and logo panel

Description

Shiny UI widgets to generate hexagon logo(s). `hexLogo()` generates a single hexagon, and `hexPanel()` generates a panel of hex logos

Usage

```

hexLogo(
  id,
  title = "",
  hex_img,
  hex_link = "",
  footer = "",
  footer_link = "",
  x = "-10",
  y = "-20",
  target_blank = FALSE
)

hexPanel(
  id,
  title,

```

```

    hex_imgs,
    hex_links = NULL,
    hex_titles = NULL,
    footers = NULL,
    footer_links = NULL,
    xs = NULL,
    ys = NULL,
    target_blank = FALSE
)

```

Arguments

id	input ID
title	title of the logo, display on top of logo or title of logo panel displayed on the left
hex_img	single value of hex_imgs
hex_link	single value of hex_links
footer	single value of footers
footer_link	single value of footer_links
x	number, X offset, e.g. "-10" instead of -10L
y	number, Y offset
target_blank	bool, whether to add target="_blank" to the link?
hex_imgs	a character vector of logo image source, can be online or local, see details
hex_links	a character vector of links attached to each logo, if not NULL, must be the same length as hex_imgs
hex_titles	similar to hex_links, titles of each logo
footers	a character vector of footer attached to each logo
footer_links	a character vector of footer links, if not NULL, must be the same length as footers
xs	a character vector X coordinate offset value for each logo image, default -10, must be the same length as hex_imgs
ys	Y coordinates offset, must be the same length as xs, default -20

Details

The image in each hexagon is resized to the same size as the hex border and then enlarged 125%. You may want to use x, y offset value to change the image position.

If your image source is local, you need to add your local directory to the shiny server, e.g. `addResourcePath("sps", "www")`. This example add www folder under my current working directory as sps to the server. Then you can access my images by `hex_imgs = "sps/my_img.png"`.

some args in `hexPanel` are character vectors, use NULL for the default value. If you want to change value but not all of your logos, use "" to occupy space in the vector. e.g. I have 3 logos, but I only want to add 2 footer and only 1 footer has a link: `footers = c("footer1", "footer2", "")`, `footer_links = c("", "https://mylink", "")`. By doing so `footers` and `footer_links` has the same required length.

Value

HTML elements

Examples

```

if(interactive()){
  ui <- fluidPage(
    hexLogo(
      "logo", "Logo",
      hex_img = "https://live.staticflickr.com/7875/46106952034_954b8775fa_b.jpg",
      hex_link = "https://www.google.com",
      footer = "Footer",
      footer_link = "https://www.google.com"
    ),
    hexLogo(
      "x", "Change X offset",
      hex_img = "https://live.staticflickr.com/7875/46106952034_954b8775fa_b.jpg",
      x = "40"
    ),
    hexLogo(
      "y", "Change Y offset",
      hex_img = "https://live.staticflickr.com/7875/46106952034_954b8775fa_b.jpg",
      y = "-60"
    ),
    hexPanel(
      "demo1", "basic panel:" ,
      rep("https://live.staticflickr.com/7875/46106952034_954b8775fa_b.jpg", 2)
    ),
    hexPanel(
      "demo2", "panel with links:" ,
      c(paste0("https://d33wubrfki0168.cloudfront.net/",
        "2c6239d311be6d037c251c71c3902792f8c4ddd2/12f67/css/images/hex/ggplot2.png"),
        paste0("https://d33wubrfki0168.cloudfront.net/",
        "621a9c8c5d7b47c4b6d72e8f01f28d14310e8370/193fc/css/images/hex/dplyr.png")
      ),
      c("https://ggplot2.tidyverse.org/", "https://dplyr.tidyverse.org/"),
      c("ggplot2", "dplyr")
    ),
    hexPanel(
      "demo3", "footer with links:" ,
      rep("https://live.staticflickr.com/7875/46106952034_954b8775fa_b.jpg", 2),
      footers = c("hex1", "hex2"),
      footer_links = rep("https://www.google.com", 2)
    ),
    hexPanel(
      "demo4", "panel offsets" ,
      hex_imgs = rep("https://live.staticflickr.com/7875/46106952034_954b8775fa_b.jpg", 4),
      footers = paste0("hex", 1:4),
      ys = seq(-20, -50, by = -10),
      xs = seq(20, 50, by = 10)
    )
  )
}

```



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

hrefTab

Display a list of links in a row of buttons

Description

hrefTab creates a small section of link buttons

Usage

```
hrefTab(
  label_texts,
  hrefs,
  Id = NULL,
  title = "A list of tabs",
  title_color = "#0275d8",
  bg_colors = "#337ab7",
  text_colors = "white",
  target_blank = FALSE,
  ...
)
```

Arguments

label_texts	individual tab labels
hrefs	individual tab links
Id	optional element ID
title	element title
title_color	title color
bg_colors	individual tab button background color, either 1 value to apply for all of them or specify for each of them in a vector
text_colors	individual tab button text color, either 1 value to apply for all of them or specify for each of them in a vector
target_blank	bool, whether to add target="_blank" to the link?
...	other arguments to be passed to the html element

Details

1. label_texts, hrefs must be the same length
2. If more than one value is provided for bg_colors or/and text_colors, the length of these 2 vectors must be the same as label_texts
3. Use "" to occupy the space if you do not want a label contains a link, e.g hrefs = c("https://google.com/", "", "")
4. If a label does not have a link, you cannot click it and there is no hovering effects.

Value

a Shiny component

Examples

```
if(interactive()){
  ui <- fluidPage(
    hrefTab(
      title = "Default",
      label_texts = c("Bar Plot", "PCA Plot", "Scatter Plot"),
      hrefs = c("https://google.com/", "", "")
    ),
    hrefTab(
      title = "Different background",
      label_texts = c("Bar Plot", "PCA Plot", "Scatter Plot"),
      hrefs = c("https://google.com/", "", ""),
      bg_colors = c("#eee", "orange", "green")
    ),
    hrefTab(
      title = "Different background and text colors",
      label_texts = c("Bar Plot", "Disabled", "Scatter Plot"),
      hrefs = c("https://google.com/", "", ""),
      bg_colors = c("green", "#eee", "orange"),
      text_colors = c("#caffc1", "black", "blue")
    )
  )
}

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

shinyApp(ui, server)
}
```

Description

creates a table in Shiny which the cells are hyper reference (links) buttons. This function is similar to [hrefTab](#), but that function only creates a single row of link buttons, and this function creates a table of rows.

The table has two columns, the first column is row names, second column is different link buttons.

Usage

```
hrefTable(
  item_titles,
  item_labels,
  item_hrefs,
  item_title_colors = "#0275d8",
  item_bg_colors = "#337ab7",
  item_text_colors = "white",
  Id = NULL,
  first_col_name = "Category",
  second_col_name = "Options",
  title = "A Table buttons with links",
  title_color = "#0275d8",
  target_blank = FALSE,
  ...
)
```

Arguments

<code>item_titles</code>	vector of strings, a vector of titles for table row names
<code>item_labels</code>	list, a list of character vectors to specify button labels in each table row, one vector per row
<code>item_hrefs</code>	list, a list of character vectors to specify button hrefs links in each table row, one vector per row
<code>item_title_colors</code>	a single character value or a character vector to specify button title text colors of each row name
<code>item_bg_colors</code>	a single character value or a list, a list of character vectors to specify button background colors in each table row, one vector per row
<code>item_text_colors</code>	a single character value or a list, a list of character vectors to specify button text colors in each table row, one vector per row
<code>Id</code>	optional ID
<code>first_col_name</code>	first column name
<code>second_col_name</code>	second column name
<code>title</code>	title of this table
<code>title_color</code>	table title color
<code>target_blank</code>	bool, whether to add <code>target="_blank"</code> to the link?
<code>...</code>	other HTML param you want to pass to the table

Details

1. `item_titles`, `item_labels`, `item_hrefs` must have the same length. Each vector in `item_labels`, `item_hrefs` must also have the same length. For example, if we want to make a table of two rows, the first row has 1 cell and the second row has 2 cells:

```
hrefTable(
  item_titles = c("row 1", "row 2"),
  item_labels = list(c("cell 1"), c("cell 1", "cell 2")),
  item_hrefs = list(c("link1"), c("link1", "link2"))
)
```

1. If `item_title_colors`, `item_text_colors` are given more than one value, the list must have the same length as `item_titles`, and length of each vector in the list must match the vector in `item_labels` in the same order.
2. If `item_title_colors` is given more than one value, the vector must have the same length as `item_titles`.
3. Use `" "` to occupy the space if you do not want a label contains a link, e.g `item_hrefs = list(c("https://www.google.com/"), c("", ""))`
4. If a label does not have a link, you cannot click it and there is no hovering effects.

Value

HTML elements

Examples

```
if(interactive()){
  ui <- fluidPage(
    hrefTable(
      title = "default",
      item_titles = c("workflow 1", "unclickable"),
      item_labels = list(c("tab 1"), c("tab 3", "tab 4")),
      item_hrefs = list(c("https://www.google.com/"), c("", ""))
    ),
    hrefTable(
      title = "Change button color and text color",
      item_titles = c("workflow 1", "No links"),
      item_labels = list(c("tab 1"), c("tab 3", "tab 4")),
      item_hrefs = list(c("https://www.google.com/"), c("", "")),
      item_bg_colors = list(c("blue"), c("red", "orange")),
      item_text_colors = list(c("black"), c("yellow", "green"))
    ),
    hrefTable(
      title = "Change row name colors and width",
      item_titles = c("Green", "Red", "Orange"),
      item_labels = list(c("tab 1"), c("tab 3", "tab 4"), c("tab 5", "tab 6", "tab 7")),
      item_hrefs = list(
        c("https://www.google.com/"),
        c("", ""),
        c("https://www.google.com/", "https://www.google.com/", "")
      )
    )
  )
}
```

```

    ),
    item_title_colors = c("green", "red", "orange"),
    style = "width: 50%"
  )
)

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

}

shinyApp(ui, server)
}

```

loadDF

Load tabular files as tibbles to server

Description

load a file to server end. It's designed to be used with a input file source switch button. It uses `vroom::vroom` to load the file. In SPS, this function is usually combined as downstream of `dynamicFileServer()` function on on the server side to read the file into R. This loading function only works for parsing tabular data, use `vroom::vroom()` internally.

If no user data is uploaded, it will return the example dataset that is prepared by the developer. If the developer does not provide the dataset either, it will return a 8-row empty tibble.

Usage

```

loadDF(
  choice,
  data_init = NULL,
  upload_path = NULL,
  eg_path = NULL,
  comment = "#",
  delim = "\t",
  col_types = vroom::cols(),
  ...
)

```

Arguments

choice	where this file comes from, one of 'upload' or example 'eg'?
data_init	a tibble to return if upload_path or eg_path is not provided. Return a 8x8 empty tibble if not provided
upload_path	when choice is "upload", where to load the file, will return data_init if this param is not provided

eg_path	when choice is "eg", where to load the file, will return data_init if this param is not provided
comment	comment characters to parse the datafile, see help file of vroom::vroom
delim	delimiter characters to parse the data file, see help file of vroom::vroom
col_types	columns specifications, see help file of vroom::vroom
...	other params for vroom, see help file of vroom::vroom

Details

This function is wrapped by the [shinyCatch\(\)](#) function, so it will show loading information both on console and on UI. This function prevents loading file errors to crash the Shiny app, so any kind of file upload will not crash the app. To show message on UI, `spsDepend("toastr")` must be used in Shiny UI function, see examples.

Value

returns a tibble and NULL if parsing fails

Examples

```
if(interactive()){
  library(shinyWidgets)
  # change value to 'local' to see the difference
  spsOption("mode", value = "server")
  ui <- fluidPage(
    spsDepend("toastr"),
    shinyWidgets::radioGroupButtons(
      inputId = "data_source", label = "Choose your data file source:",
      selected = "upload",
      choiceNames = c("Upload", "Example"),
      choiceValues = c("upload", "eg")
    ),
    dynamicFile("data_path", label = "input file"),
    dataTableOutput("df")
  )

  server <- function(input, output, session) {
    tmp_file <- tempfile(fileext = ".csv")
    write.csv(iris, file = tmp_file)
    upload_path <- dynamicFileServer(input, session, "data_path")
    data_df <- reactive({
      loadDF(choice = input$data_source,
            upload_path = upload_path()$datapath,
            delim = ",", eg_path = tmp_file)
    })
    output$df <- renderDataTable(data_df())
  }
  shinyApp(ui, server)
}
```

pgPaneUI

*A draggable progress panel***Description**

Creates a panel that displays multiple progress items. Use [pgPaneUI](#) on UI side and use `pgPaneUpdate` to update it.

A overall progress is automatically calculated on the bottom.

Usage

```
pgPaneUI(
  pane_id,
  titles,
  pg_ids,
  title_main = NULL,
  opened = FALSE,
  top = "3%",
  right = "2%"
)
```

```
pgPaneUpdate(pane_id, pg_id, value, session = getDefaultReactiveDomain())
```

Arguments

pane_id	Progress panel main ID, use ns wrap it on pgPaneUI but not on pgPaneUpdate if using shiny module
titles	labels to display for each progress, must have the same length as pg_ids
pg_ids	a character vector of IDs for each progress. Don't forget to use ns wrap each ID.
title_main	If not specified and pane_id contains 'plot', title will be 'Plot Prepare'; has 'df' will be 'Data Prepare', if neither will be "Progress"
opened	bool, if this panel is opened at start
top	css style off set to the current window top
right	css style off set to the current window right
pg_id	a character string of ID indicating which progress within this panel you want to update. Do not use ns(pg_id) to wrap it on server
value	0-100 number to update the progress you use pg_id to choose
session	current shiny session

Value

HTML elements

Examples

```

if(interactive()){
  # try to slide c under 0
  ui <- fluidPage(
    h4("you need to open up the progress
        tracker, it is collapsed ->"),
    actionButton("a", "a"),
    actionButton("b", "b"),
    sliderInput("c", min = -100,
                max = 100, value = 0,
                label = "c"),
    pgPaneUI(
      pane_id = "thispg",
      titles = c("this a", "this b", " this c"),
      pg_ids = c("a", "b", "c"),
      title_main = "Example Progress",
      opened = TRUE,
      top = "30%",
      right = "50%"
    )
  )
  server <- function(input, output, session) {
    observeEvent(input$a, {
      for(i in 1:10){
        pgPaneUpdate("thispg", "a", i*10)
        Sys.sleep(0.3)
      }
    })
    observeEvent(input$b, {
      for(i in 1:10){
        pgPaneUpdate("thispg", "b", i*10)
        Sys.sleep(0.3)
      }
    })
    observeEvent(input$c, pgPaneUpdate("thispg", "c", input$c))
  }
  shinyApp(ui, server)
}

```

renderDesc

Render some collapsible markdown text

Description

write some text in markdown format and it will help you render to markdown, use [shiny::markdown](#) but it is collapsible.

Usage

```
renderDesc(id, desc)
```


Arguments

id	element ID
desc	one character string in markdown format

Value

HTML elements

Examples

```
if(interactive()){
  desc <-
  "
  # Some desc
  - xxxx
  - bbbb

  This is a [link](https://www.google.com/).

  `Some other things`
  > other markdown things

  1. aaa
  2. bbb
  3. ccc
  "
  ui <- fluidPage(
    renderDesc(id = "desc", desc),
  )

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

  }

  shinyApp(ui, server)
}
```

Description

Exception in Shiny apps can crash the app. Most time we don't want the app to crash but just stop this code block, inform users and continue with other code blocks. This function is designed to handle these issues.

Usage

```
shinyCatch(
  expr,
  position = "bottom-right",
  blocking_level = "none",
  shiny = TRUE,
  prefix = "SPS",
  trace_back = spsOption("traceback")
)
```

Arguments

expr	expression
position	client side message bar position, one of: c("top-right", "top-center", "top-left", "top-full-width", "bottom-right", "bottom-center", "bottom-left", "bottom-full-width").
blocking_level	what level you want to block the execution, one of "error", "warning", "message", default is "none", do not block following code execution.
shiny	bool, only show message on console log but not in Shiny app when it is FALSE. Useful if you want to keep the exception only to the server and hide from your users. You do not need to set it to FALSE when purely work outside shiny, it will automatically detect if you are working in a Shiny environment or not.
prefix	character, what prefix to display on console for the log, e.g. for error, the default will be displayed as "SPS-ERROR". You can make your own prefix, like prefix = "MY", then, it will be "MY-ERROR". Use "" if you do not want any prefix, like prefix = "", then, it will just be "ERROR". multiple levels
trace_back	bool, added since spsComps 0.2, if the expression is blocked or has errors, cat the full trace back? It will display called functions and code source file and line number if possible. Default follows the SPS spsOption("traceback") setting. You can set it by running spsOption("traceback", TRUE). If you do not set it, it will be FALSE. or you can just manually set it for each individual shinyCatch call shinyCatch({...}, trace_back = TRUE).

Details

- The blocking works similar to shiny's `shiny::req()` and `shiny::validate()`. If anything inside fails, it will block the rest of the code in your reactive expression domain.
- It will show error, warning, message by a toastr bar on client end and also log the text on server console depending on the `blocking_level` (dual-end logging).
- If blocks at error level, function will be stopped and other code in the same reactive context will be blocked.
- If blocks at warning level, warning and error will be blocked.
- message level blocks all 3 levels.
- If `blocking_level` is other than these 3, no exceptions will be block, and if there is any error, NULL will return and following code will continue to run.

- To have the message displayed on shiny, `spsDepend("toastr")` must be added in UI if you are working on your own app not under SPS framework, see examples.
- Messages will be displayed for 3s, 5s for warnings and errors will never go away on UI unless users' mouse hover on the bar or manually close it.

Value

see description and details

Examples

```
if(interactive()){
  ui <- fluidPage(
    spsDepend("toastr"),
    h4("Run this example on your own computer to better understand exception
      catch and dual-end logging", class = "text-center"),
    column(
      6,
      actionButton("btn1", "error and blocking"),
      actionButton("btn2", "error no blocking"),
      actionButton("btn3", "warning but still returns value"),
      actionButton("btn4", "warning but blocking returns"),
      actionButton("btn5", "message"),
    ),
    column(
      6,
      verbatimTextOutput("text")
    )
  )
}
server <- function(input, output, session) {
  fn_warning <- function() {
    warning("this is a warning!")
    return("warning returns")
  }
  observeEvent(input$btn1, {
    shinyCatch(stop("error with blocking"), blocking_level = "error")
    output$text <- renderPrint("You shouldn't see me")
  })
  observeEvent(input$btn2, {
    shinyCatch(stop("error without blocking"))
    output$text <- renderPrint("I am not blocked by error")
  })
  observeEvent(input$btn3, {
    return_value <- shinyCatch(fn_warning())
    output$text <- renderPrint("warning and blocked")
  })
  observeEvent(input$btn4, {
    return_value <- shinyCatch(fn_warning(), blocking_level = "warning")
    print(return_value)
    output$text <- renderPrint("other things")
  })
  observeEvent(input$btn5, {
```

```

    shinyCatch(message("some message"))
    output$text <- renderPrint("some message")
  })
}
shinyApp(ui, server)
}
# outside shiny examples
shinyCatch(message("this message"))
try({shinyCatch(stop("this error")); "no block"}, silent = TRUE)
try({shinyCatch(stop("this error"), blocking_level = "error"); "blocked"}, silent = TRUE)

```

shinyCheckPkg

Shiny package checker

Description

A server end function to check package namespace for some required packages of users' environment. If all packages are installed, a successful message will be displayed on the bottom-right. If not, pop up a message box in shiny to tell users how to install the missing packages.

This is useful when some of packages are required by a shiny app. Before running into that part of code, using this function to check the required package and pop up warnings will prevent app to crash.

Usage

```

shinyCheckPkg(
  session,
  cran_pkg = NULL,
  bioc_pkg = NULL,
  github = NULL,
  quietly = FALSE
)

```

Arguments

session	shiny session
cran_pkg	a vector of package names
bioc_pkg	a vector of package names
github	a vector of github packages, github package must use the format of "github user name/ repository name", eg. c("user1/pkg1", "user2/pkg2")
quietly	bool, should warning messages be suppressed?

Value

TRUE if pass, sweet alert message and FALSE if fail

Examples

```

if(interactive()){
  library(shiny)

  ui <- fluidPage(
    tags$label('Check if package "pkg1", "pkg2", "bioxxx",
              github package "user1/pkg1" are installed'), br(),
    actionButton("check_random_pkg", "check random_pkg"),
    br(), spsHr(),
    tags$label('We can combine `spsValidate` to block server code to prevent
              crash if some packages are not installed. '), br(),
    tags$label('If "shiny" is installed, make a plot. '), br(),
    actionButton("check_shiny", "check shiny"), br(),
    tags$label('If "ggplot99" is installed, make a plot. '), br(),
    actionButton("check_gg99", "check ggplot99"), br(),
    plotOutput("plot_pkg")
  )

  server <- function(input, output, session) {
    observeEvent(input$check_random_pkg, {
      shinyCheckPkg(session, cran_pkg = c("pkg1", "pkg2"),
                    bioc_pkg = "bioxxx", github = "user1/pkg1")
    })
    observeEvent(input$check_shiny, {
      spsValidate(verbose = FALSE, {
        if(!shinyCheckPkg(session, cran_pkg = c("shiny"))) stop("Install packages")
      })
      output$plot_pkg <- renderPlot(plot(1))
    })
    observeEvent(input$check_gg99, {
      spsValidate({
        if(!shinyCheckPkg(session, cran_pkg = c("ggplot99"))) stop("Install packages")
      })
      output$plot_pkg <- renderPlot(plot(99))
    })
  }

  shinyApp(ui, server)
}

```

spsCodeBtn

Display your code in a bootstrap modal or collapse

Description

Developers often want to show their code in a shiny app. This function creates a button that when clicked, a modal or collapse hidden element will show up to display your code.

Usage

```
spsCodeBtn(
  id,
  code,
  language = "r",
  label = "",
  title = "Code to Reproduce",
  show_span = FALSE,
  tool_tip = "Show Code",
  placement = "bottom",
  btn_icon = icon("code"),
  display = c("modal", "collapse"),
  size = c("large", "medium", "small"),
  color = "black",
  shape = c("rect", "circular"),
  ...
)
```

Arguments

id	element ID
code	code you want to display, in a character string or vector.
language	string, what programming language is the code, use shinyAce::getAceModes() to see options
label	string, label to display on the button
title	string, title of the modal or collapse
show_span	bool, use the tag to show a little label of the left of the button? The span text will use text from tool_tip
tool_tip	string, what tooltip to display when hover on the button
placement	string, where to display the tooltip
btn_icon	icon, shiny::icon() , icon of the button
display	string, one of "modal", "collapse"
size	string, one of "large", "medium", "small", only works for modal
color	string, color of the button
shape	string, shape of the button, one of "rect", "circular",
...	other args pass to the shiny::actionButton

Details

1. The modal or collapse has an ID, the ID is your button ID + "-modal" or "-collapse", like "my_button-modal"
2. You could update the code inside the collapse use [shinyAce::updateAceEditor](#) on server, the code block ID is button ID + "-ace", like "my_button-ace" . See examples.

Value

a shiny element

Examples

```
if(interactive()){
  library(shiny)
  my_code <-
  ,
  # load package and data
  library(ggplot2)
  data(mpg, package="ggplot2")
  # mpg <- read.csv("http://goo.gl/uEeRGu")

  # Scatterplot
  theme_set(theme_bw()) # pre-set the bw theme.
  g <- ggplot(mpg, aes(cty, hwy))
  g + geom_jitter(width = .5, size=1) +
    labs(subtitle="mpg: city vs highway mileage",
         y="hwy",
         x="cty",
         title="Jittered Points")
  ,
html_code <-
  ,
  <!DOCTYPE html>
  <html>
  <body>

  <h2>ABC</h2>

  <p id="demo">Some HTML</p>

  </body>
  </html>
  ,
ui <- fluidPage(
  fluidRow(
    column(
      6,
      h3("Display by modal"),
      column(
        6, h4("default"),
        spsCodeBtn(id = "a", my_code)
      ),
      column(
        6, h4("change color and shape"),
        spsCodeBtn(
          id = "b", c(my_code, my_code),
          color = "red", shape = "circular")
        )
    ),
  ),
```

```

column(
  6,
  h3("Display by collapse"),
  column(
    6, h4("collapse"),
    spsCodeBtn(id = "c", my_code, display = "collapse")
  ),
  column(
    6, h4("different programming language"),
    spsCodeBtn(
      id = "d", html_code,
      language = "html", display = "collapse")
  )
),
fluidRow(
  column(
    6,
    h3("Update code"),
    spsCodeBtn(
      "update-code",
      "# No code here",
      display = "collapse"
    ),
    actionButton("update", "change code in the left `spsCodeBtn`"),
    actionButton("changeback", "change it back")
  )
)
)

server <- function(input, output, session) {
  observeEvent(input$update, {
    shinyAce::updateAceEditor(
      session, editorId = "update-code-ace",
      value = "# code has changed!\n 1+1"
    )
  })
  observeEvent(input$changeback, {
    shinyAce::updateAceEditor(
      session, editorId = "update-code-ace",
      value = "# No code here"
    )
  })
}

shinyApp(ui, server)
}

```


Description

Mostly used in SPS internal development or add dependencies for some server end functions. For most UI functions, the dependency has been automatically attached for you.

Usage

```
spsDepend(dep, js = TRUE, css = TRUE, listing = TRUE)
```

Arguments

dep	dependency names, see details
js	bool, use only javascript from this resource if there are both js and css files?
css	bool, use only CSS from this resource if there are both js and css files?
listing	bool, if your dep is invalid, list all options? FALSE will mute it.

Details

For dep, current options are:

- basic: spsComps basic css and js
- css_loading: for css loaders
- update_pg: spsComps [pgPaneUpdate](#) function required, js only
- update_timeline: spsComps [spsTimeline](#) function required, js only
- font-awesome: font-awesome, css only
- btn: comes from shinyWidgets package, btn.css, css only
- toastr: comes from shinytoastr package, toastr.js, css and js
- pop-tip: enable enhanced bootstrap popover and tips, required for [bsHoverPopover](#) function
- gotop: required by [spsGoTop](#) function
- animation: required for animation related functions to add animations for icons and other elements
- css-loader: required for loader functions

Value

[htmltools::htmlDependency](#) object

Examples

```
# list all options
spsDepend("")
# try some options
spsDepend("basic")
spsDepend("font-awesome")
# Then add it to your shiny app
if(interactive()){
  library(shiny)
```

```

ui <- fluidPage(
  tags$i(class = "fa fa-home"),
  spsDepend("font-awesome")
)

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

}

shinyApp(ui, server)
}

```

spsGoTop

Go top button

Description

add a go top button on your shiny app. When the user clicks the button, scroll the window all the way to the top. Just add this function anywhere in you UI.

Usage

```

spsGoTop(
  id = "gotop",
  icon = NULL,
  right = "1rem",
  bottom = "10rem",
  color = "#337ab7"
)

```

Arguments

id	element ID
icon	shiny::icon if you do not want to use the default rocket image
right	character string, css style, the button's position to window right
bottom	character string, css style, the button's position to window bottom
color	color of the icon.

Details

The button hides if you are on very top of the page. If you scroll down 50px, this button will appear.

Value

a shiny component

Examples

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

  ui <- fluidPage(
    h1("Scroll the page..."),
    lapply(1: 100, function(x) br()),
    spsGoTop("default"),
    spsGoTop("mid", right = "50%", bottom= "50%", icon = icon("home"), color = "red"),
    spsGoTop("up", right = "95%", bottom= "95%", icon = icon("arrow-up"), color = "green")
  )

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

  }

  shinyApp(ui, server)
}
```

spsHr

hr line with color #3b8dbc38

Description

hr line with color #3b8dbc38

Usage

```
spsHr()
```

Value

HTML <hr> element

Examples

```
spsHr()
```

spsTimeline

A shiny timeline component

Description

This timeline is horizontal, use **spsTimeline** to define it and use **updateSpsTimeline** on server to update it.

Usage

```
spsTimeline(id, up_labels, down_labels, icons, completes)
```

```
updateSpsTimeline(
  session,
  id,
  item_no,
  complete = TRUE,
  up_label = NULL,
  down_label = NULL
)
```

Arguments

id	html ID of the timeline if you are using shiny modules: use namespace function to create the ID but DO NOT use namespace function on server.
up_labels	a vector of strings, text you want to display on top of each timeline item, usually like year number. If you do not want any text for a certain items, use "" to occupy the space.
down_labels	a vector of strings, text you want to display at the bottom of each timeline item.
icons	a list of icon objects. If you do not want an icon for certain items, use div() to occupy the space.
completes	a vector of TRUE or FALSE, indicating if the items are completed or not. Completed items will become green.
session	current shiny session
item_no	integer, which item number counting from left to right you want to update
complete	bool, is this item completed or not
up_label	the item_no associated up label to update
down_label	the item_no associated down label to update

Details

up_labels, down_labels, icons, completes must have the same length.

Value

returns a shiny component

Examples

```

if(interactive()){
  ui <- fluidPage(
    column(6,
      spsTimeline(
        "b",
        up_labels = c("2000", "2001"),
        down_labels = c("step 1", "step 2"),
        icons = list(icon("table"), icon("gear")),
        completes = c(FALSE, TRUE)
      )
    ),
    column(6,
      actionButton("a", "complete step 1"),
      actionButton("c", "uncomplete step 1"))
  )

  server <- function(input, output, session) {
    observeEvent(input$a, {
      updateSpsTimeline(session, "b", 1, up_label = "0000", down_label = "Finish")
    })
    observeEvent(input$c, {
      updateSpsTimeline(session, "b", 1, complete = FALSE,
        up_label = "9999", down_label = "Step 1")
    })
  }

  shinyApp(ui, server)
}

```

spsValidate

Validate expressions

Description

this function is used on server side to usually validate input dataframe or some expression. The usage is similar to [shiny::validate](#) but is not limited to shiny render functions and provides better user notification and server-end logging (dual-end logging).

Usage

```

spsValidate(
  expr,
  vd_name = "my validation",

```

```

pass_msg = glue("validation: '{vd_name}' passed"),
shiny = TRUE,
verbose = spsOption("verbose"),
prefix = ""
)

```

Arguments

expr	the expression to validate data or other things. Use <code>stop("your message")</code> or generate some errors inside to fail the validation. If there is no error, it will return TRUE and display <code>pass_msg</code> on both console and shiny app if <code>verbose = TRUE</code> or global SPS option <code>verbose</code> is TRUE. If the expression fails, it will block the code following this function within the same reactive domain to continue, similar to shinyCatch() .
vd_name	validate title
pass_msg	string, if pass, what message do you want to show
shiny	bool, show message on console but hide from users? see shinyCatch() for more details
verbose	bool, show pass message? Default follows global verbose setting, use spsUtil::spsOption to set up the value <code>spsOption("verbose, TRUE")</code> to turn on and <code>spsOption("verbose, FALSE")</code> to turn off and <code>spsOption("verbose")</code> to check current setting, see examples.
prefix	see prefix in shinyCatch()

Details

- To have the message displayed on shiny, `spsDepend("toastr")` must be added in UI if you are working on you own app not under SPS framework, see examples.

Value

If expression fails, block the code following this validation function and no final return, else TRUE.

Examples

```

if(interactive()){
  ui <- fluidPage(
    spsDepend("toastr"), # need to add toastr dependency
    column(
      4,
      h3("click below to make the plot"),
      p("this button will succeed, verbose on"),
      actionButton("vd1", "make plot 1"),
      plotOutput("p1")
    ),
    column(
      4,
      h3("click below to make the plot"),
      p("this button will succeed, verbose off"),

```

```

        actionButton("vd2", "make plot 2"),
        plotOutput("p2")
    ),
    column(
        4,
        h3("click below to make the plot"),
        p("this button will fail, no plot will be made"),
        actionButton("vd3", "make plot 3"),
        plotOutput("p3")
    ),
    column(
        4,
        h3("click below to make the plot"),
        p("this button will fail, but the message is hidden from users"),
        actionButton("vd4", "make plot 4"),
        plotOutput("p4")
    )
)
)
server <- function(input, output, session) {
  mydata <- datasets::iris
  observeEvent(input$vd1, {
    spsOption("verbose", TRUE) # use global sps verbose setting
    spsValidate({
      is.data.frame(mydata)
    }, vd_name = "Is dataframe")
    output$p1 <- renderPlot(plot(iris$Sepal.Length, iris$Sepal.Width))
  })
  observeEvent(input$vd2, {
    spsValidate({
      is.data.frame(mydata)
    },
    vd_name = "Is dataframe",
    verbose = FALSE) # use in-function verbose setting
    output$p2 <- renderPlot(plot(iris$Sepal.Length, iris$Sepal.Width))
  })
  observeEvent(input$vd3, {
    spsValidate({
      is.data.frame(mydata)
      if(nrow(mydata) <= 200) stop("Input needs more than 200 rows")
    })
    print("other things blocked")
    output$p3 <- renderPlot(plot(iris$Sepal.Length, iris$Sepal.Width))
  })
  observeEvent(input$vd4, {
    spsValidate({
      is.data.frame(mydata)
      if(nrow(mydata) <= 200) stop("Input needs more than 200 rows")
    }, shiny = FALSE)
    print("other things blocked")
    output$p4 <- renderPlot(plot(iris$Sepal.Length, iris$Sepal.Width))
  })
}
shinyApp(ui, server)

```

```
}  
# outside shiny example  
mydata2 <- list(a = 1, b = 2)  
spsValidate({mydata2}), "Not empty")  
try(spsValidate(stopifnot(is.data.frame(mydata2)), "is dataframe?"), silent = TRUE)
```

tabTitle	<i>h2 title with bootstrap info color</i>
----------	---

Description

Mostly used under SPS framework to create the tab title.

Usage

```
tabTitle(title, ...)
```

Arguments

title	title text
...	other attributes and children to this element

Value

a h2 level heading with bootstrap4 "info" color(bt4 color not the default bt3 info color)

Examples

```
tabTitle("This title")
```

textButton	<i>Text input with an action button</i>
------------	---

Description

One kind of bootstrap3 input group: a textinput and a button attached to the end

Usage

```
textButton(
  textId,
  btnId = paste0(textId, "_btn"),
  label = "",
  text_value = "",
  placeholder = "",
  tooltip = "",
  placement = "bottom",
  btn_icon = NULL,
  btn_label = "btn",
  style = "",
  ...
)
```

Arguments

textId	the text input ID
btnId	the button ID, if not specified, it is "textId" + "_btn" like, textId_btn
label	label of the whole group, on the top
text_value	initial value of the text input
placeholder	placeholder text of the text input
tooltip	a tooltip of the group
placement	where should the tooltip go?
btn_icon	a shiny::icon of the button
btn_label	text on the button
style	additional CSS style of the group
...	additional args pass to the button, see shiny::actionButton

Value

a shiny input group

Examples

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

  ui <- fluidPage(
    column(
      6,
      textButton(textId = "tbtn_default", label = "default"),
      textButton(
        textId = "tbtn-icon",
        label = "change icon and color",
        btn_icon = icon("home"),
```

```

      class = "btn-warning" # pass to the button
    ),
    textButton(
      textId = "tbtn_style",
      label = "change styles",
      style = "color: red; border: 2px dashed green;"
    ),
    textButton(
      textId = "tbtn_submit",
      label = "interact with shiny server",
      btn_label = "Submit",
      placeholder = "type and submit",
      class = "btn-primary"),
    verbatimTextOutput("tbtn_submit_out")
  )
)

server <- function(input, output, session) {
  # watch for the button ID "tbtn_submit" + "_btn"
  observeEvent(input$tbtn_submit_btn, {
    output$tbtn_submit_out <- renderPrint(isolate(input$tbtn_submit))
  })
}

shinyApp(ui, server)
}

```

textInputGroup

Bootstrap 3 text input group

Description

Text input group and custom widgets append to left ar/and right

Usage

```

textInputGroup(
  textId,
  label = "",
  value = "",
  placeholder = "enter text",
  left_text = NULL,
  right_text = NULL,
  style = "width: 100%;"
)

```

Arguments

textId	text box id
label	text label for this input group
value	default value for the text input
placeholder	default placeholder text for the text input if no value
left_text	text or icon add to the left side
right_text	text or icon add to the right side
style	additional style add to the group

Details

If no text is specified for both left and right, the return is almost identical to [clearableTextInput](#)

Value

text input group component

Examples

```
if(interactive()){  
  
  ui <- fluidPage(  
    textInputGroup("id1", "left", left_text = "a"),  
    textInputGroup("id2", "right", right_text = "b"),  
    textInputGroup("id3", "both", left_text = "$", right_text = ".00"),  
    textInputGroup("id4", "none"),  
    textInputGroup("id5", "icon", left_text = icon("home")),  
  )  
  
  server <- function(input, output, session) {  
  
  }  
  
  shinyApp(ui, server)  
}
```

Index

addLoader, 2
animateIcon, 7
animateServer, 9, 10
animateServer (animateUI), 9
animateUI, 9, 9
animationRemove, 9, 10
animationRemove (animateUI), 9

bsHoverPopover, 12, 41
bsplus::bs_embed_popover, 12
bsplus::bs_set_data(), 12

clearableTextInput, 13, 51
cssLoader, 14

dynamicFile, 16
dynamicFileServer (dynamicFile), 16
dynamicFileServer(), 29

gallery, 19

heightMatcher, 21
hexLogo, 22
hexLogo(), 22
hexPanel (hexLogo), 22
hexPanel(), 22
hrefTab, 25, 27
hrefTable, 26
htmltools::htmlDependency, 41
htmltools::tag, 12

loadDF, 29

pgPaneUI, 31, 31
pgPaneUpdate, 41
pgPaneUpdate (pgPaneUI), 31

renderDesc, 32

shiny::actionButton, 15, 38, 49
shiny::actionButton(), 12

shiny::fileInput, 16
shiny::icon, 7, 8, 42, 49
shiny::icon(), 38
shiny::markdown, 32
shiny::req(), 34
shiny::validate, 45
shiny::validate(), 34
shinyAce::getAceModes(), 38
shinyAce::updateAceEditor, 38
shinyCatch, 33
shinyCatch(), 30, 46
shinyCheckPkg, 36
shinyFiles, 17
spsCodeBtn, 37
spsDepend, 40
spsGoTop, 41, 42
spsHr, 43
spsTimeline, 41, 44
spsUtil::spsOption, 46
spsUtil::spsOption(), 18
spsValidate, 45

tabTitle, 48
textButton, 48
textInputGroup, 50

updateSpsTimeline (spsTimeline), 44

vroom::vroom, 29, 30
vroom::vroom(), 29