Package: shiny386 (via r-universe)

May 28, 2026

Title Old School 'Bootstrap 4' Template for Shiny

Version 0.0.1.9000

Description This template is made for people having nostalgic feelings about floppy disks and other lovely stuff from that time. It is built on top of the Bootstrap 386 template <https://github.com/kristopolous/BOOTSTRA.386>. Less distraction for more productivity!

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

URL https://github.com/RinteRface/shiny386, https://rinterface.github.io/shiny386

BugReports https://github.com/RinteRface/shiny386/issues

Imports htmltools (>= 0.5.2), shiny, httpuv

Suggests bslib

Depends R (>= 2.10)

Config/pak/sysreqs cmake make libuv1-dev zlib1g-dev

Repository https://cynkra.r-universe.dev

Date/Publication 2024-09-26 19:20:47 UTC

RemoteUrl https://github.com/RinteRface/shiny386

RemoteRef HEAD

RemoteSha 2b6e18fbb94f8ead09975438ea16a3da9bcf1cd3

Contents

`badge_386` *Create a Bootstrap 386 badge*

Description

Create a Bootstrap 386 badge

Usage

```
badge_386(..., status, rounded = FALSE)
```

Arguments

...	Text.
status	Badge status.
rounded	Rounded style. Default to FALSE.

Value

A shiny tags

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)
 ui <- page_386(
  badge_386(status = "danger", "1"),
  badge_386(status = "info", "2"),
  badge_386(status = "success", "3", rounded = TRUE)
 )

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

---

button_386 *Bootstrap 386 action button*

Description

Bootstrap 386 action button

Usage

```
button_386(inputId, label, status = NULL, icon = NULL, width = NULL, ...)
```

Arguments

inputId	The input slot that will be used to access the value.
label	The contents of the button or link–usually a text label, but you could also use any other HTML, like an image.
status	Button color.
icon	An optional icon() to appear on the button.
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
...	Named attributes to be applied to the button or link.

Value

A shiny tag.

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(
  button_386(
   "btn",
   HTML(paste("Value", textOutput("val"), sep = ":")),
   icon = icon("thumbs-up"),
   class = "btn-lg"
  )
 )

 server <- function(input, output) {
   output$val <- renderText(input$btn)
 }

 shinyApp(ui, server)
}
```

---

card_386 *Create a Bootstrap 386 card*

Description

Create a Bootstrap 386 card

Create a Bootstrap 4 card title element

Create a Bootstrap 386 card subtitle element

Create a Bootstrap 386 card link element

Usage

```
card_386(..., title = NULL, status = NULL, footer = NULL)

card_title_386(title)

card_subtitle_386(subtitle)

card_link_386(href, label)
```

Arguments

...	Card content.
title	Title text.
status	Card background status.
footer	Card footer.
subtitle	Card subtitle.
href	Target url.
label	Link text.

Value

A shiny tag

A shiny tag.

A shiny tag.

A shiny tag.

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)
 ui <- page_386(
  card_386(
   title = "My card",
   "This is my card",
   br(),
   card_link_386(href = "https://www.google.com", "More"),
   footer = "Card footer"
  )
 )

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

---

checkbox_group_input_386

Create a Bootstrap 386 checkbox group input

Description

Create a Bootstrap 386 checkbox group input

Usage

```
checkbox_group_input_386(
  inputId,
  label,
  choices = NULL,
  selected = NULL,
  width = NULL,
  choiceNames = NULL,
  choiceValues = NULL
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
choices	List of values to show checkboxes for. If elements of the list are named then that name rather than the value is displayed to the user. If this argument is provided, then choiceNames and choiceValues must not be provided, and vice-versa. The values should be strings; other types (such as logicals and numbers) will be coerced to strings.
selected	The values that should be initially selected, if any.
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
choiceNames, choiceValues	List of names and values, respectively, that are displayed to the user in the app and correspond to the each choice (for this reason, choiceNames and choiceValues must have the same length). If either of these arguments is provided, then the other *must* be provided and choices *must not* be provided. The advantage of using both of these over a named list for choices is that choiceNames allows any type of UI object to be passed through (tag objects, icons, HTML code, ...), instead of just simple text. See Examples.

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(
  checkbox_group_input_386("variable", "Variables to show:",
   c("Cylinders" = "cyl",
     "Transmission" = "am",
     "Gears" = "gear")),
  tableOutput("data")
 )

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

```
  }, rownames = TRUE)
 }
 shinyApp(ui, server)

}
```

---

checkbox_input_386 *Create a Bootstrap 386 checkbox*

Description

Create a Bootstrap 386 checkbox

Usage

```
checkbox_input_386(inputId, label, value = FALSE, width = NULL)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
value	Initial value (TRUE or FALSE).
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(
  checkbox_input_386("check", "Check me", TRUE),
  verbatimTextOutput("val")
 )

 server <- function(input, output, session) {
   output$val <- renderPrint(input$check)
 }
 shinyApp(ui, server)
}
```

create_checkbox_tag *Create checkbox/switch input based on the selected type*

Description

Used internally by toggle_input_386 and checkbox_input_386

Usage

```
create_checkbox_tag(
  inputId,
  label,
  value = FALSE,
  width = NULL,
  type = c("switch", "checkbox")
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
value	Initial value (TRUE or FALSE).
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
type	Input type. This is to be able to distinguish between switch and checkbox, which have slightly different design.

Value

An input tag.

dropdown_386 *Create a Bootstrap 386 dropdown container for buttons/links*

Description

Can be a simple link or an action button

Usage

```
dropdown_386(..., inputId = NULL, label, status = NULL, open = FALSE)

open_dropdown_386(inputId, session = shiny::getDefaultReactiveDomain())

dropdown_item_386(inputId = NULL, href = NULL, label)
```

Arguments

...	Slot for dropdown_item_386.
inputId	If action button.
label	Button label.
status	Button status.
open	Whether to open the dropdown at start. Default to FALSE.
session	Shiny session object.
href	If simple link.

Value

A shiny tag

A message from R to JavaScript through the websocket.

A shiny tag

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(
   dropdown_386(
     inputId = "plop",
     label = "Menu",
     status = "danger",
     open = FALSE,
     dropdown_item_386(inputId = "btn1", label = "button 1"),
     dropdown_item_386(href = "https://www.google.com/", label = "More")
   )
 )

 server <- function(input, output, session) {
   observe(print(input$btn1))
 }
 shinyApp(ui, server)
}
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(
  fluidRow(
    button_386("open", "Open dropdown", class = "btn-lg"),
    dropdown_386(
      inputId = "plop",
      label = "Menu",
      dropdown_item_386(inputId = "btn1", label = "button 1"),
      dropdown_item_386(href = "https://www.google.com/", label = "More")
```

```
    )
  )
 )

 server <- function(input, output, session) {
   observe(print(input$plop))
   observeEvent(input$open, {
     open_dropdown_386("plop")
   })
   observeEvent(req(input$plop), {
     showNotification("Dropdown opened!")
   })
 }
 shinyApp(ui, server)
}
```

`jumbotron_386` *Create a Bootstrap 386 Jumbotron*

Description

Create a Bootstrap 386 Jumbotron

Usage

```
jumbotron_386(..., title = NULL)
```

Arguments

`...`	Any element.
`title`	Jumbotron title.

Value

A shiny tag

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(
  jumbotron_386(
   title = "Jumbotron 386",
   p("Hello World"),
   button_386("btn", "More", class = "btn-lg btn-block")
  )
 )
```

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

 shinyApp(ui, server)
}
```

`list_group_386` *Create a Bootstrap 386 list group container*

Description

Create a Bootstrap 386 list group container

Usage

```
list_group_386(..., width = 4)
```

Arguments

`...`	Slot for list_group_item_386.
`width`	List group width. 4 by default. Between 1 and 12.

Author(s)

David Granjon, <dgranjon@ymail.com>

Examples

```
if(interactive()){
 library(shiny)
 library(shiny386)

 shinyApp(
   ui = page_386(
      fluidRow(
       list_group_386(
       list_group_item_386(
        type = "basic",
        "Cras justo odio"
       ),
       list_group_item_386(
        type = "basic",
        "Dapibus ac facilisis in"
       ),
       list_group_item_386(
        type = "basic",
        "Morbi leo risus"
       )
      ),
```

```
      list_group_386(
       list_group_item_386(
        "Cras justo odio",
        active = TRUE,
        disabled = FALSE,
        type = "action",
        src = "http://www.google.fr"
       ),
       list_group_item_386(
        active = FALSE,
        disabled = FALSE,
        type = "action",
        "Dapibus ac facilisis in",
        src = "http://www.google.fr"
       ),
       list_group_item_386(
        "Morbi leo risus",
        active = FALSE,
        disabled = TRUE,
        type = "action",
        src = "http://www.google.fr"
       )
      ),
      list_group_386(
       list_group_item_386(
        "Donec id elit non mi porta gravida at eget metus.
        Maecenas sed diam eget risus varius blandit.",
        active = TRUE,
        disabled = FALSE,
        type = "heading",
        title = "List group item heading",
        subtitle = "3 days ago",
        footer = "Donec id elit non mi porta."
       ),
       list_group_item_386(
        "Donec id elit non mi porta gravida at eget metus.
        Maecenas sed diam eget risus varius blandit.",
        active = FALSE,
        disabled = FALSE,
        type = "heading",
        title = "List group item heading",
        subtitle = "3 days ago",
        footer = "Donec id elit non mi porta."
       )
      )
     )
   ),
   server = function(input, output) {}
 )
}
```

`list_group_item_386` *Create a Bootstrap 386 list group item*

Description

Create a Bootstrap 386 list group item

Usage

```
list_group_item_386(
  ...,
  active = FALSE,
  disabled = FALSE,
  type = c("basic", "action", "heading"),
  src = "#",
  title = NULL,
  subtitle = NULL,
  footer = NULL
)
```

Arguments

`...`	Item content.
`active`	Whether the item is active or not. FALSE by default. Only if type is "action" or "heading".
`disabled`	Whether the item is disabled or not. FALSE by default. Only if type is "action" or "heading".
`type`	Item type. Choose between "basic", "action" and "heading".
`src`	Item external link.
`title`	Item title (only if type is "heading").
`subtitle`	Item subtitle (only if type is "heading").
`footer`	Item footer content (only if type is "heading").

Author(s)

David Granjon, <dgranjon@ymail.com>

modal_386 *Create a Bootstrap 386 modal*

Description

Create a Bootstrap 386 modal

Show a Bootstrap 386 modal

Hide a Bootstrap 386 modal

Usage

```
modal_386(
  ...,
  title = NULL,
  footer = modalButton("Dismiss"),
  size = c("m", "s", "l", "xl"),
  easyClose = FALSE,
  fade = TRUE
)

show_modal_386(ui, session = getDefaultReactiveDomain())

remove_modal_386(session = getDefaultReactiveDomain())
```

Arguments

...	UI elements for the body of the modal dialog box.
title	An optional title for the dialog.
footer	UI for footer. Use NULL for no footer.
size	One of "s" for small, "m" (the default) for medium, "l" for large, or "xl" for extra large. Note that "xl" only works with Bootstrap 4 and above (to opt-in to Bootstrap 4+, pass bslib::bs_theme() to the theme argument of a page container like fluidPage()).
easyClose	If TRUE, the modal dialog can be dismissed by clicking outside the dialog box, or be pressing the Escape key. If FALSE (the default), the modal dialog can't be dismissed in those ways; instead it must be dismissed by clicking on a modalButton(), or from a call to removeModal() on the server.
fade	If FALSE, the modal dialog will have no fade-in animation (it will simply appear rather than fade in to view).
ui	UI content to show in the modal.
session	The session object passed to function given to shinyServer.

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 shinyApp(
  ui = page_386(
    button_386("show", "Show modal dialog"),
    verbatimTextOutput("dataInfo")
  ),

  server = function(input, output) {
    # reactiveValues object for storing current data set.
    vals <- reactiveValues(data = NULL)

   # Return the UI for a modal dialog with data selection input. If 'failed' is
   # TRUE, then display a message that the previous value was invalid.
   dataModal <- function(failed = FALSE) {
     modal_386(
       textInput("dataset", "Choose data set",
                 placeholder = 'Try "mtcars" or "abc"'
       ),
       span('(Try the name of a valid data object like "mtcars", ',
            'then a name of a non-existent object like "abc")'),
       if (failed)
         div(tags$b("Invalid name of data object", style = "color: red;")),

       footer = tagList(
         modalButton("Cancel"),
         button_386("ok", "OK")
       )
     )
   }

   # Show modal when button is clicked.
   observeEvent(input$show, {
     show_modal_386(dataModal())
   })

   # When OK button is pressed, attempt to load the data set. If successful,
   # remove the modal. If not show another modal, but this time with a failure
   # message.
   observeEvent(input$ok, {
     # Check that data object exists and is data frame.
     if (!is.null(input$dataset) && nzchar(input$dataset) &&
         exists(input$dataset) && is.data.frame(get(input$dataset))) {
       vals$data <- get(input$dataset)
       remove_modal_386()
     } else {
       show_modal_386(dataModal(failed = TRUE))
     }
   })
```

```
   # Display information about selected data
   output$dataInfo <- renderPrint({
     if (is.null(vals$data))
       "No data selected"
     else
       summary(vals$data)
   })
  }
 )
}
```

---

navbar_page_386 *Create a Bootstrap 386 navbar page*

Description

Create a Bootstrap 386 navbar page

Create a Bootstrap 386 navbar menu

Update a Bootstrap 386 navbar on the client

Usage

```
navbar_page_386(
  title,
  ...,
  id = NULL,
  selected = NULL,
  position = c("static-top", "fixed-top", "fixed-bottom"),
  header = NULL,
  footer = NULL,
  inverse = FALSE,
  windowTitle = title
)

navbar_menu_386(title, ..., menuName = title, icon = NULL)

update_navbar_page_386(
  session = getDefaultReactiveDomain(),
  inputId,
  selected = NULL
)
```

Arguments

title	The title to display in the navbar

...	tabPanel() elements to include in the page. The navbarMenu function also accepts strings, which will be used as menu section headers. If the string is a set of dashes like "----" a horizontal separator will be displayed in the menu.
id	If provided, you can use input$id in your server logic to determine which of the current tabs is active. The value will correspond to the value argument that is passed to tabPanel().
selected	The value (or, if none was supplied, the title) of the tab that should be selected by default. If NULL, the first tab will be selected.
position	Determines whether the navbar should be displayed at the top of the page with normal scrolling behavior ("static-top"), pinned at the top ("fixed-top"), or pinned at the bottom ("fixed-bottom"). Note that using "fixed-top" or "fixed-bottom" will cause the navbar to overlay your body content, unless you add padding, e.g.: tags$style(type="text/css", "body {padding-top: 70px;}")
header	Tag or list of tags to display as a common header above all tabPanels.
footer	Tag or list of tags to display as a common footer below all tabPanels
inverse	TRUE to use a dark background and light text for the navigation bar
windowTitle	the browser window title (as a character string). The default value, NA, means to use any character strings that appear in title (if none are found, the host URL of the page is displayed by default).
menuName	A name that identifies this navbarMenu. This is needed if you want to insert/remove or show/hide an entire navbarMenu.
icon	Optional icon to appear on a navbarMenu tab.
session	The session object passed to function given to shinyServer. Default is getDefaultReactiveDomain()
inputId	The id of the tabsetPanel, navlistPanel, or navbarPage object.

Value

A shiny tag

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- navbar_page_386(
  "App Title",
  id = "tabset",
  tab_panel_386(
    "Tab 1",
    radio_input_386(
      "dist", "Distribution type:",
      c("Normal" = "norm",
        "Uniform" = "unif",
        "Log-normal" = "lnorm",
        "Exponential" = "exp")
    ),
```

```
    plotOutput("distPlot")
   ),
   tab_panel_386(
    "Tab 2",
    select_input_386(
     "variable", "Variable:",
     c("Cylinders" = "cyl",
      "Transmission" = "am",
      "Gears" = "gear")
    ),
    tableOutput("data")
   ),
   navbar_menu_386(
    "More",
    tab_panel_386("Summary", "Extra content 1"),
    "----",
    "Section header",
    tab_panel_386("Table", "Extra content 2")
   )
  )

  server <- function(input, output, session) {
   output$distPlot <- renderPlot({
    dist <- switch(input$dist,
                   norm = rnorm,
                   unif = runif,
                   lnorm = rlnorm,
                   exp = rexp,
                   rnorm)

    hist(dist(500))
   })

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

   observe(print(input$tabset))

  }
  shinyApp(ui, server)
 }
 if (interactive()) {
  library(shiny)
  library(shiny386)

  ui <- navbar_page_386(
   "App Title",
   id = "tabset",
   selected = "Tab 2",
   header = radio_input_386("controller", "Update tab", 1:4),
   tabPanel(
    "Tab 1",
```

```
    "Content 1"
  ),
  tabPanel(
    "Tab 2",
    "Content 2"
  ),
  navbar_menu_386(
    "More",
    tab_panel_386("Tab 3", "Extra content 1"),
    "----",
    "Section header",
    tab_panel_386("Tab 4", "Extra content 2")
  )
 )

 server <- function(input, output, session) {
   observeEvent(input$controller, {
     update_navbar_page_386(session, "tabset",
                            selected = paste("Tab", input$controller)
     )
   })
 }
 shinyApp(ui, server)

}
```

---

page_386 *Create Bootstrap 386 page skeleton*

Description

Create Bootstrap 386 page skeleton

Usage

```
page_386(..., title = NULL)
```

Arguments

...	Slot for shiny386 layout elements.
title	The browser window title (defaults to the host URL of the page). Can also be set as a side effect of the titlePanel() function.

Value

A list of tags

progress_386 *Create a Bootstrap 386 progress bar*

Description

The progress bar may be updated server side. See update_progress_386.

Usage

```
progress_386(id = NULL, value, status = NULL)
```

Arguments

id	Progress unique id.
value	Progress value. Numeric between 0 and 100.
status	Progress status.

Value

A progress bar tag.

See Also

update_progress_386

radio_input_386 *Create a Bootstrap 386 radio buttons*

Description

Create a Bootstrap 386 radio buttons

Usage

```
radio_input_386(
  inputId,
  label,
  choices = NULL,
  selected = NULL,
  width = NULL,
  choiceNames = NULL,
  choiceValues = NULL
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
choices	List of values to select from (if elements of the list are named then that name rather than the value is displayed to the user). If this argument is provided, then choiceNames and choiceValues must not be provided, and vice-versa. The values should be strings; other types (such as logicals and numbers) will be coerced to strings.
selected	The initially selected value. If not specified, then it defaults to the first item in choices. To start with no items selected, use character(0).
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
choiceNames, choiceValues	List of names and values, respectively, that are displayed to the user in the app and correspond to the each choice (for this reason, choiceNames and choiceValues must have the same length). If either of these arguments is provided, then the other *must* be provided and choices *must not* be provided. The advantage of using both of these over a named list for choices is that choiceNames allows any type of UI object to be passed through (tag objects, icons, HTML code, ...), instead of just simple text. See Examples.

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(
  radio_input_386("dist", "Distribution type:",
                  c("Normal" = "norm",
                    "Uniform" = "unif",
                    "Log-normal" = "lnorm",
                    "Exponential" = "exp")),
  plotOutput("distPlot")
 )

 server <- function(input, output, session) {
   output$distPlot <- renderPlot({
     dist <- switch(input$dist,
                    norm = rnorm,
                    unif = runif,
                    lnorm = rlnorm,
                    exp = rexp,
                    rnorm)

     hist(dist(500))
   })
 }
 shinyApp(ui, server)

}
```

`select_input_386` *Create a Bootstrap 386 select input*

Description

Create a Bootstrap 386 select input

Usage

```
select_input_386(
  inputId,
  label,
  choices,
  selected = NULL,
  multiple = FALSE,
  selectize = FALSE,
  width = NULL,
  size = NULL
)
```

Arguments

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

Note

Incompatible with selectize. Set to FALSE by default to have correct CSS rendering.

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(
  select_input_386("variable", "Variable:",
   c("Cylinders" = "cyl",
     "Transmission" = "am",
     "Gears" = "gear")),
   tableOutput("data")
 )

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

}
```

---

shiny386 *shiny386*

Description

Old school Bootstrap 4 template for Shiny

Author(s)

Maintainer: David Granjon <dgranjon@ymail.com>

See Also

Useful links:

- https://github.com/RinteRface/shiny386
- https://rinterface.github.io/shiny386
- Report bugs at https://github.com/RinteRface/shiny386/issues

show_toast_386 *Show a Bootstrap 386 toast on the client*

Description

Show a Bootstrap 386 toast on the client

Usage

```
show_toast_386(id, options = NULL, session = getDefaultReactiveDomain())
```

Arguments

id	Toast id.
options	Toast options: see https://getbootstrap.com/docs/4.3/components/toasts/.
session	Shiny session

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(
  toast_386(
    id = "toast",
    title = "Hello",
    subtitle = "now",
    "Toast body",
    img = "https://preview-dev.tabler.io/static/logo.svg"
  ),
  button_386("launch", "Go!", class = "btn-lg")
 )

 server <- function(input, output, session) {
   observe(print(input$toast))
   observeEvent(input$launch, {
     removeNotification("notif")
     show_toast_386(
       "toast",
       options = list(
         animation = FALSE,
         delay = 3000
       )
     )
   })

   observeEvent(input$toast, {
     showNotification(
```

```
        id = "notif",
        "Toast was closed",
        type = "warning",
        duration = 1,

      )
    })
  }

  shinyApp(ui, server)
}
```

`tab_panel_386` *Create a Bootstrap 386 tab panel*

Description

Create a Bootstrap 386 tab panel

Usage

```
tab_panel_386(title, ..., value = title, icon = NULL)
```

Arguments

`title`	Display title for tab
`...`	UI elements to include within the tab
`value`	The value that should be sent when `tabsetPanel` reports that this tab is selected. If omitted and `tabsetPanel` has an `id`, then the title will be used.
`icon`	Optional icon to appear on the tab. This attribute is only valid when using a `tabPanel` within a `navbarPage()`.

`tabset_panel_386` *Create a Bootstrap 386 tabset panel*

Description

Create a Bootstrap 386 tabset panel

Usage

```
tabset_panel_386(
  ...,
  id = NULL,
  selected = NULL,
  type = c("tabs", "pills"),
  position = NULL
)
```

Arguments

...	tabPanel() elements to include in the tabset
id	If provided, you can use input$id in your server logic to determine which of the current tabs is active. The value will correspond to the value argument that is passed to tabPanel().
selected	The value (or, if none was supplied, the title) of the tab that should be selected by default. If NULL, the first tab will be selected.
type	"tabs" Standard tab look "pills" Selected tabs use the background fill color "hidden" Hides the selectable tabs. Use type = "hidden" in conjunction with tabPanelBody() and updateTabsetPanel() to control the active tab via other input controls. (See example below)
position	Tabs position (left or right).

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(
  tabset_panel_386(
    id = "tabset",
    selected = "Tab 2",
    tabPanel("Tab 1", "Content 1"),
    tabPanel("Tab 2", "Content 2")
  )
 )

 server <- function(input, output, session) {
   observe(print(input$tabset))
 }
 shinyApp(ui, server)

}
```

---

text_area_input_386 *Create a Bootstrap 386 text area input*

Description

Create a Bootstrap 386 text area input

Usage

```
text_area_input_386(
  inputId,
  label,
  value = "",
  width = NULL,
  height = NULL,
  cols = NULL,
  rows = NULL,
  placeholder = NULL,
  resize = NULL
)
```

Arguments

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

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(
  textAreaInput("caption", "Caption", "Data Summary"),
  verbatimTextOutput("value")
 )

 server <- function(input, output, session) {
  output$value <- renderText({ input$caption })
 }
 shinyApp(ui, server)
```

```
}
```

`text_input_386` *Create a Bootstrap 386 text input*

Description

Create a Bootstrap 386 text input

Usage

```
text_input_386(inputId, label, value = "", width = NULL, placeholder = NULL)
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`label`	Display label for the control, or `NULL` for no label.
`value`	Initial value.
`width`	The width of the input, e.g. `'400px'`, or `'100%'`; see `validateCssUnit()`.
`placeholder`	A character string giving the user a hint as to what can be entered into the control. Internet Explorer 8 and 9 do not support this option.

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(
  textInput("caption", "Caption", "Data Summary"),
  verbatimTextOutput("value")
 )

 server <- function(input, output, session) {
  output$value <- renderText({ input$caption })
 }
 shinyApp(ui, server)

}
```

toast_386 *Create a Bootstrap 386 toast*

Description

Display user feedback

Usage

```
toast_386(id, title = NULL, subtitle = NULL, ..., img = NULL)
```

Arguments

id	Unique toast id.
title	Toast title.
subtitle	Toast subtitle.
...	Toast content.
img	Toast image.

Value

A toast

See Also

show_toast_386

toggle_input_386 *Custom Bootstrap 386 switch input*

Description

Similar to the shiny checkbox

Usage

```
toggle_input_386(inputId, label, value = FALSE, width = NULL)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
value	Initial value (TRUE or FALSE).
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().

Value

A toggle input tag.

See Also

update_toggle_input_386.

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(
  toggle_input_386("toggle", "Toggle me", TRUE),
  verbatimTextOutput("val")
 )

 server <- function(input, output, session) {
   output$val <- renderPrint(input$toggle)
 }
 shinyApp(ui, server)
}
```

---

update_checkbox_group_input_386

Change the value of a checkbox group input on the client

Description

Change the value of a checkbox group input on the client

Usage

```
update_checkbox_group_input_386(
  session,
  inputId,
  label = NULL,
  choices = NULL,
  selected = NULL,
  choiceNames = NULL,
  choiceValues = NULL
)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
choices	List of values to show checkboxes for. If elements of the list are named then that name rather than the value is displayed to the user. If this argument is provided, then choiceNames and choiceValues must not be provided, and vice-versa. The values should be strings; other types (such as logicals and numbers) will be coerced to strings.
selected	The values that should be initially selected, if any.
choiceNames, choiceValues	List of names and values, respectively, that are displayed to the user in the app and correspond to the each choice (for this reason, choiceNames and choiceValues must have the same length). If either of these arguments is provided, then the other *must* be provided and choices *must not* be provided. The advantage of using both of these over a named list for choices is that choiceNames allows any type of UI object to be passed through (tag objects, icons, HTML code, ...), instead of just simple text. See Examples.

See Also

checkbox_group_input_386()

Examples

```
if (interactive()) {

 ui <- page_386(
   p("The first radio button group controls the second"),
   checkbox_group_input_386("inCheckboxGroup", "Input radio buttons",
     c("Item A", "Item B", "Item C")),
   checkbox_group_input_386("inCheckboxGroup2", "Input radio buttons 2",
     c("Item A", "Item B", "Item C"))
 )

 server <- function(input, output, session) {
   observe({
    x <- input$inCheckboxGroup

    # Can use character(0) to remove all choices
    if (is.null(x))
      x <- character(0)

    # Can also set the label and select items
    update_checkbox_group_input_386(session, "inCheckboxGroup2",
                            label = paste("Checkboxgroup label", length(x)),
                            choices = x,
                            selected = x
    )
```

```
  })
 }

 shinyApp(ui, server)
}
```

---

update_checkbox_input_386

Update checkbox_input_386 on the client

Description

Update checkbox_input_386 on the client

Usage

```
update_checkbox_input_386(session, inputId, label = NULL, value = NULL)
```

Arguments

session	The session object passed to function given to shinyServer. Default is getDefaultReactiveDomain()
inputId	The id of the input object.
label	The label to set for the input object.
value	Initial value (TRUE or FALSE).

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(
  button_386("update", "Go!", class = "btn-lg"),
  checkbox_input_386("check", "Checked", value = TRUE)
 )

 server <- function(input, output, session) {
   observe(print(input$check))
   observeEvent(input$update, {
     update_checkbox_input_386(
       session,
       "toggle",
       value = !input$check
     )
   })
 }

 shinyApp(ui, server)
}
```

update_progress_386 *Update a progress_386 on the client*

Description

Update a progress_386 on the client

Usage

```
update_progress_386(id, value, session = shiny::getDefaultReactiveDomain())
```

Arguments

id	Progress unique id.
value	New value.
session	Shiny session object.

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)
 ui <- page_386(
  button_386("update", "Update"),
  br(),
  progress_386(id = "progress1", 12)
 )

 server <- function(input, output, session) {
   observeEvent(input$update, {
     update_progress_386(
       id = "progress1",
       sample(1:100, 1)
     )
   })
 }
 shinyApp(ui, server)
}
```

update_radio_input_386 *Change the value of a radio input on the client*

Description

Change the value of a radio input on the client

Usage

```
update_radio_input_386(
  session,
  inputId,
  label = NULL,
  choices = NULL,
  selected = NULL,
  choiceNames = NULL,
  choiceValues = NULL
)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
choices	List of values to select from (if elements of the list are named then that name rather than the value is displayed to the user). If this argument is provided, then choiceNames and choiceValues must not be provided, and vice-versa. The values should be strings; other types (such as logicals and numbers) will be coerced to strings.
selected	The initially selected value. If not specified, then it defaults to the first item in choices. To start with no items selected, use character(0).
choiceNames, choiceValues	List of names and values, respectively, that are displayed to the user in the app and correspond to the each choice (for this reason, choiceNames and choiceValues must have the same length). If either of these arguments is provided, then the other *must* be provided and choices *must not* be provided. The advantage of using both of these over a named list for choices is that choiceNames allows any type of UI object to be passed through (tag objects, icons, HTML code, ...), instead of just simple text. See Examples.

See Also

radio_input_386()

Examples

```
if (interactive()) {

 ui <- page_386(
   p("The first radio button group controls the second"),
   radio_input_386("inRadioButtons", "Input radio buttons",
     c("Item A", "Item B", "Item C")),
   radio_input_386("inRadioButtons2", "Input radio buttons 2",
     c("Item A", "Item B", "Item C"))
 )
```

```
 server <- function(input, output, session) {
   observe({
     x <- input$inRadioButtons

    # Can also set the label and select items
    update_radio_input_386(session, "inRadioButtons2",
      label = paste("radioButtons label", x),
      choices = x,
      selected = x
    )
  })
 }

 shinyApp(ui, server)
}
```

---

update_select_input_386 *Update a Bootstrap 386 select input on the client*

Description

Update a Bootstrap 386 select input on the client

Usage

```
update_select_input_386(
  session = getDefaultReactiveDomain(),
  inputId,
  label = NULL,
  choices = NULL,
  selected = NULL
)
```

Arguments

session	The session object passed to function given to shinyServer. Default is getDefaultReactiveDomain()
inputId	The id of the input object.
label	The label to set for the input object.
choices	List of values to select from. If elements of the list are named, then that name — rather than the value — is displayed to the user. It's also possible to group related inputs by providing a named list whose elements are (either named or unnamed) lists, vectors, or factors. In this case, the outermost names will be used as the group labels (leveraging the <optgroup> HTML tag) for the elements in the respective sublist. See the example section for a small demo of this feature.
selected	The initially selected value (or multiple values if multiple = TRUE). If not specified then defaults to the first value for single-select lists and no values for multiple select lists.

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(
  p("The radio group controls the select input"),
  radio_input_386("inCheckboxGroup", "Input checkbox",
                    c("Item A", "Item B", "Item C")),
  select_input_386("inSelect", "Select input",
              c("Item A", "Item B", "Item C"))
 )

 server <- function(input, output, session) {
   observe({
     x <- input$inCheckboxGroup

     # Can use character(0) to remove all choices
     if (is.null(x))
       x <- character(0)

     # Can also set the label and select items
     update_select_input_386(session, "inSelect",
                       label = paste("Select input label", length(x)),
                       choices = x,
                       selected = tail(x, 1)
     )
   })
 }

 shinyApp(ui, server)
}
```

`update_tabset_panel_386` *Update a Bootstrap 386 tabset panel on the client*

Description

Update a Bootstrap 386 tabset panel on the client

Usage

```
update_tabset_panel_386(
  session = getDefaultReactiveDomain(),
  inputId,
  selected = NULL
)
```

Arguments

session	The session object passed to function given to shinyServer. Default is getDefaultReactiveDomain()
inputId	The id of the tabsetPanel, navlistPanel, or navbarPage object.
selected	The value (or, if none was supplied, the title) of the tab that should be selected by default. If NULL, the first tab will be selected.

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(sidebarLayout(
 sidebarPanel(
   radio_input_386("controller", "Controller", choices = c(1, 2, 3))
 ),
 mainPanel(
   tabset_panel_386(id = "inTabset",
               tab_panel_386(title = "Panel 1", value = "panel1", "Panel 1 content"),
               tab_panel_386(title = "Panel 2", value = "panel2", "Panel 2 content"),
               tab_panel_386(title = "Panel 3", value = "panel3", "Panel 3 content")
   )
 )
 ))

 server <- function(input, output, session) {
   observeEvent(input$controller, {
     update_tabset_panel_386(session, "inTabset",
                       selected = paste0("panel", input$controller)
     )
   })
 }

 shinyApp(ui, server)
}
```

---

update_text_area_input_386

Update a Bootstrap 386 text area input on the client

Description

Update a Bootstrap 386 text area input on the client

Usage

```
update_text_area_input_386(
  session = getDefaultReactiveDomain(),
```

```
  inputId,
  label = NULL,
  value = NULL,
  placeholder = NULL
)
```

Arguments

session	The session object passed to function given to shinyServer. Default is getDefaultReactiveDomain()
inputId	The id of the input object.
label	The label to set for the input object.
value	Initial value.
placeholder	A character string giving the user a hint as to what can be entered into the control. Internet Explorer 8 and 9 do not support this option.

---

update_text_input_386 *Update a Bootstrap 386 text input on the client*

Description

Update a Bootstrap 386 text input on the client

Usage

```
update_text_input_386(
  session = getDefaultReactiveDomain(),
  inputId,
  label = NULL,
  value = NULL,
  placeholder = NULL
)
```

Arguments

session	The session object passed to function given to shinyServer. Default is getDefaultReactiveDomain()
inputId	The id of the input object.
label	The label to set for the input object.
value	Initial value.
placeholder	A character string giving the user a hint as to what can be entered into the control. Internet Explorer 8 and 9 do not support this option.

update_toggle_input_386 *Update toggle_input_386 on the client*

Description

Update toggle_input_386 on the client

Usage

```
update_toggle_input_386(session, inputId, label = NULL, value = NULL)
```

Arguments

session	The session object passed to function given to shinyServer. Default is getDefaultReactiveDomain()
inputId	The id of the input object.
label	The label to set for the input object.
value	Initial value (TRUE or FALSE).

Examples

```
if (interactive()) {
 library(shiny)
 library(shiny386)

 ui <- page_386(
  button_386("update", "Go!", class = "btn-lg"),
  toggle_input_386("toggle", "Switch", value = TRUE)
 )

 server <- function(input, output, session) {
   observe(print(input$toggle))
   observeEvent(input$update, {
     update_toggle_input_386(
       session,
       "toggle",
       value = !input$toggle
     )
   })
 }

 shinyApp(ui, server)
}
```

use_bs4_deps *Create shiny386 dependencies*

Description

Add all necessary dependencies so that shiny386 renders well

Usage

```
use_bs4_deps(tag)
```

Arguments

tag	Tag on which to add dependencies. We usually target the body.

See Also

page_386.

validate_progress_value *Validate a progress_386 value*

Description

Validate a progress_386 value

Usage

```
validate_progress_value(value)
```

Arguments

value	Value to validate.

Value

An error is raised if the value does not met expectations.

validate_status *Validation function*

Description

Validate the status of a Bootstrap 386 element.

Usage

```
validate_status(status)
```

Arguments

status	Color to validate.

Value

TRUE if the test pass.

Examples

```
## Not run:
 validate_status("danger")
 validate_status("maroon")

## End(Not run)
```

Index