Package: gridstackeR (via r-universe)

September 9, 2024

Type Package

Title Wrapper for 'gridstack.js'

Version 0.1.1

Maintainer Peter Gandenberger <peter.gandenberger@gmail.com>

Description An easy way to create responsive layouts with just a few lines of code. You can create boxes that are draggable and resizable and load predefined Layouts. The package serves as a wrapper to allow for easy integration of the 'gridstack.js' functionalities <https://github.com/gridstack/gridstack.js>.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Depends R (>= 3.5.0)

Imports htmltools, shiny, shinyjs, checkmate

Suggests shinydashboard, shinytest2

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

RemoteUrl https://github.com/petergandenberger/gridstacker

RemoteRef HEAD

RemoteSha 6b7dff430b16fccf82740a69166f8dada1e1aac7

Contents

`check_grid_stack_item_list` *Checks that all given arguments are* `grid_stack_item`*s*

Description

Checks that all given arguments are `grid_stack_items`

Usage

```
check_grid_stack_item_list(...)
```

Arguments

`...`	arguments to be checked

Value

TRUE if arguments are valid `grid_stack_items`, FALSE otherwise

`gridstackeR_demo` *Demo*

Description

a short example of gridstackeR

Usage

```
gridstackeR_demo()
```

Value

an example shiny shinyApp that uses the gridstackeR package to create a responsive layout with resizable and draggable boxes.

Examples

```
## Not run:
gridstackeR_demo()

## End(Not run)
```

grid_stack *Grid Stack Container*

Description

This acts as a container for the grid_stack_item's.

Usage

```
grid_stack(
  ...,
  id = "",
  opts = "{cellHeight: 70}",
  ncols = 12,
  nrows = 12,
  dynamic_full_window_height = FALSE,
  height_offset = 0
)
```

Arguments

`...`	all `grid_stack_items` contained in this grid. No arguments other than `grid_stack_items` are allowed here.
`id`	the id of the grid_stack container used for multi-grid layouts. (if no id is provided, a random id is generated)
`opts`	grid options: check gridstack documentation for more details
`ncols`	number of columns for the grid (If you need > 12 columns you need to generate the CSS manually)
`nrows`	number of rows for the grid
`dynamic_full_window_height`	if TRUE, the grid will change dynamically to fit the window size minus the `height_offset`
`height_offset`	margin for the grid height, see `dynamic_full_window_height`

Value

a grid_stack that can contain resizable and draggable `grid_stack_items`

Examples

```
## Not run:
library(gridstackeR)
library(shiny)
library(shinydashboard)
library(shinyjs)
```

```
ui <- dashboardPage(
  title = "gridstackeR Demo",
  dashboardHeader(),
  dashboardSidebar(disable = TRUE),
  dashboardBody(
    useShinyjs(),
    # make sure the content fills the given height
    tags$style(".grid-stack-item-content {height:100%;}"),
    grid_stack(
      dynamic_full_window_height = TRUE,
      grid_stack_item(
        h = 2, w = 2,
        box(
          title = "gridstackeR", status = "success", solidHeader = TRUE,
          width = 12, height = "100%",
          div("Drag and scale the Boxes as desired")
        )
      ),
      grid_stack_item(
        h = 4, w = 4, id = "plot_container",
        box(
          title = "Histogram", status = "primary", solidHeader = TRUE,
          width = 12, height = "100%",
          plotOutput("plot", height = "auto")
        )
      ),
      grid_stack_item(
        h = 3, w = 4, min_h = 3, max_h = 3, id = "slider",
        box(
          title = "Inputs", status = "warning", solidHeader = TRUE,
          width = 12, height = "100%",
          sliderInput("slider", "Slider input:", 1, 100, 50)
        )
      ),
      grid_stack_item(
        w = 4, h = 10, x = 0, y = 0, id = "c_table",
        DT::dataTableOutput("mytable")
      )
    )
  )
)

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

  output$plot <- renderPlot({
    x    <- faithful$waiting
    bins <- seq(min(x), max(x), length.out = input$slider + 1)

    hist(x, breaks = bins, col = "#75AADB", border = "white",
         xlab = "Waiting time to next eruption (in mins)",
         main = "Histogram of waiting times")
  },
  # set the height according to the container height (minus the margins)
  height = function() {max(input$plot_container_height - 80, 150)}
  )

  output$mytable <- DT::renderDataTable({
    DT::datatable(mtcars, options = list(
      # set the height according to the container height (minus the margins)
      scrollY = max(input$c_table_height, 200) - 110, paging = FALSE
    )
    )
  })
}

shinyApp(ui, server)


## End(Not run)
```

`grid_stack_item` *Grid Stack Item*

Description

This is a wrapper for the individual items to be displayed in the grid_stack Check the gridstack documentation for more information.

The default for all parameters is an empty string, this will make them disappear for gridstackjs

Usage

```
grid_stack_item(
  ...,
  id = NULL,
  auto_position = NULL,
  x = NULL,
  y = NULL,
  w = NULL,
  h = NULL,
  max_w = NULL,
  min_w = NULL,
  max_h = NULL,
  min_h = NULL,
  locked = NULL,
  no_resize = NULL,
  no_move = NULL,
  resize_handles = NULL,
  hide_overflow = TRUE
)
```

Arguments

`...`	content to include in the grid stack item
`id`	the id of the item, used for save and load functions, this param is propagated through to lower levels. If the id is provided, changes made to the item by the user will trigger reactive inputs for width, height (both in pixels), x, y, w, h (all 4 in number of columns/rows) (see Documentation for more information)
`auto_position`	if set to TRUE x and y attributes are ignored and the element is placed to the first available position. Having either x or y missing will also do that
`x`, `y`	element position in columns/rows. Note: if one is missing this will `auto_position` the item
`w`, `h`	element size in columns/rows
`max_w`, `min_w`, `max_h`, `min_h`	element constraints in column/row (default none)
`locked`	means another widget wouldn't be able to move it during dragging or resizing. The widget can still be dragged or resized by the user. You need to add `no_resize` and `no_move` attributes to completely lock the widget.
`no_resize`	if set to TRUE it disables element resizing
`no_move`	if set to TRUE it disables element moving
`resize_handles`	- widgets can have their own custom resize handles.
`hide_overflow`	hides the overflow of the item-content by default (i.e adds "overflow:hidden;" to the style) For example 'e,w' will make that particular widget only resize east and west.

Value

a grid_stack_item to be placed inside a `grid_stack`. This item is resizable and draggable by default.

Examples

```
## Not run:
grid_stack_item(
h = 2, w = 2,
box(
  title = "gridstackeR", status = "success", solidHeader = TRUE, width = 12, height = "100%",
   div("Drag and scale the Boxes as desired")
)
)

## End(Not run)
```

Index