Package 'jsTreeR'

January 16, 2021

Type Package

Title A Wrapper of the JavaScript Library 'jsTree'

Version 1.2.0

Description Creates interactive trees that can be included in 'Shiny' apps and R markdown documents. A tree allows to represent hierarchical data (e.g. the contents of a directory). Similar to the 'shinyTree' package but offers more features and options, such as the grid extension, restricting the drag-and-drop behavior, and settings for the search functionality. It is possible to attach some data to the nodes of a tree and then to get these data in 'Shiny' when a node is selected. Also provides a 'Shiny' gadget allowing to manipulate one or more folders.

License GPL-3

URL https://github.com/stla/jsTreeR

BugReports https://github.com/stla/jsTreeR/issues

Encoding UTF-8

LazyData true

Imports htmlwidgets, shiny, htmltools, rstudioapi, shinyAce, miniUI, tools, stats, base64enc, utils, R.utils

Suggests jsonlite, magrittr

RoxygenNote 7.1.1

Depends R (>= 2.10)

NeedsCompilation no

Author Stéphane Laurent [aut, cre],
jQuery contributors [ctb, cph] (jQuery),
Ivan Bozhanov [ctb, cph] (jsTree),
Vedran Opacic [ctb, cph] (jsTree bootstrap theme),
Avi Deitcher [ctb, cph] (jsTreeGrid),
Philip Hutchison [ctb, cph] (PDFObject),
Terence Eden [ctb, cph] (SuperTinyIcons)

Maintainer Stéphane Laurent <laurent_step@outlook.fr>

Repository CRAN

Date/Publication 2021-01-16 09:30:02 UTC

R topics documented:

`Countries` *Countries*

Description

Countries data with country code, name, currency code, population, capital and continent name.

Usage

```
Countries
```

Format

A dataframe with 250 rows and 6 columns.

`folderGadget` *Folder gadget*

Description

Shiny gadget allowing to manipulate one or more folders.

Usage

```
folderGadget(
  dirs = ".",
  tabs = FALSE,
  recursive = TRUE,
  all.files = FALSE,
  trash = FALSE
)
```

Arguments

`dirs`	character vector of paths to some folders
`tabs`	logical, whether to display the trees in tabs; this option is effective only when there are two folders in the `dirs` argument
`recursive, all.files`	options passed to `list.files`; even if `all.files = TRUE`, `'.git'` and `'.Rproj.user'` folders are always discarded
`trash`	logical, whether to add a trash to the gadget, allowing to restore the files or folders you delete

Note

You can run the gadget for the current directory from the Addins menu within RStudio ('Explore current folder').

Examples

```
library(jsTreeR)

# copy a folder to a temporary location for the illustration:
tmpDir <- tempdir()
folder <- file.path(tmpDir, "htmlwidgets")
htmlwidgets <- system.file("htmlwidgets", package = "jsTreeR")
R.utils::copyDirectory(htmlwidgets, folder)
# we use a copy because the actions performed in the gadget are
# actually executed on the files system!

# explore and manipulate the folder (drag-and-drop, right-click):
if(interactive()){
  folderGadget(folder)
}

# the 'trash' option allows to restore the elements you delete:
if(interactive()){
  folderGadget(folder, trash = TRUE)
}

# you can open several folders:
folder1 <- file.path(folder, "lib")
folder2 <- file.path(folder, "gadget")
if(interactive()){
  folderGadget(c(folder1, folder2))
}
```

jstree *HTML widget displaying an interactive tree*

Description

Create a HTML widget displaying an interactive tree.

Usage

```
jstree(
  nodes,
  elementId = NULL,
  checkboxes = FALSE,
  search = FALSE,
  searchtime = 250,
  dragAndDrop = FALSE,
  dnd = NULL,
  multiple = TRUE,
  types = NULL,
  sort = FALSE,
  unique = FALSE,
  wholerow = FALSE,
  contextMenu = FALSE,
  checkCallback = NULL,
  grid = NULL,
  theme = "default"
)
```

Arguments

`nodes` data, a list of nodes; each node is a list with a required field `text`, a character string labeling the node, and optional fields

- `children` a list of nodes
- `data` a named list of data to attach to the node; see the Shiny examples
- `icon` space-separated HTML class names defining an icon, e.g. `"glyphicon glyphicon-flash"`; in a Shiny app you can also use a super tiny icon, e.g. `"supertinyicon-julia"`; see the Shiny example showing all available such icons
- `type` a character string for usage with the `types` option; see first example
- `state` a named list defining the state of the node, with four possible fields, each being `TRUE` or `FALSE`:
  - `opened` whether the node should be initially opened
  - `selected` whether the node should be initially selected
  - `disabled` whether the node should be disabled
  - `checked` whether the node should be initially checked, effective only when the `checkboxes` option is `TRUE`

`a_attr` a named list of attributes for the node label, such as `list(title = "I'm a tooltip",style = "color: red;")`

`li_attr` a named list of attributes for the whole node, including its children, such as `list(title = "I'm a tooltip",style = "background-color: pink;")`

`elementId` a HTML id for the widget (useless for common usage)

`checkboxes` logical, whether to enable checkboxes next to each node; this makes easier the selection of multiple nodes

`search` either a logical value, whether to enable the search functionality with default options, or a named list of options for the search functionality; see the Shiny example and the jsTree API documentation for the list of possible options

`searchtime` currently ignored

`dragAndDrop` logical, whether to allow the rearrangement of the nodes by dragging and dropping

`dnd` a named list of options related to the drag-and-drop functionality, e.g. the `is_draggable` function to define which nodes are draggable; see the first example and the jsTree API documentation for the list of possible options

`multiple` logical, whether to allow multiselection

`types` a named list of node properties; see first example

`sort` logical, whether to sort the nodes

`unique` logical, whether to ensure that no node label is duplicated

`wholerow` logical, whether to highlight whole selected rows

`contextMenu` either a logical value, whether to enable a context menu to create/rename/delete/cut/copy/paste nodes, or a list of options; see the jsTree API documentation for the possible options

`checkCallback` either `TRUE` to allow to perform some actions such as creating a new node, or a JavaScript function; see the example where this option is used to define restrictions on the drag-and-drop behavior

`grid` list of settings for the grid; see the second example, the Shiny example, and github.com/deitch/jstree-grid for the list of all available options

`theme` jsTree theme, one of `"default"`, `"default-dark"`, or `"proton"`

Examples

```
# example illustrating the 'dnd' and 'checkCallback' options ####

library(jsTreeR)

nodes <- list(
  list(
    text = "RootA",
    type = "root",
    children = list(
      list(
        text = "ChildA1",
        type = "child"
```

```
      ),
      list(
        text = "ChildA2",
        type = "child"
      )
    )
  ),
  list(
    text = "RootB",
    type = "root",
    children = list(
      list(
        text = "ChildB1",
        type = "child"
      ),
      list(
        text = "ChildB2",
        type = "child"
      )
    )
  )
)

types <- list(
  root = list(
    icon = "glyphicon glyphicon-ok"
  ),
  child = list(
    icon = "glyphicon glyphicon-file"
  )
)

checkCallback <- JS(
  "function(operation, node, parent, position, more) {",
  "  if(operation === 'move_node') {",
  "    if(parent.id === '#' || parent.type === 'child') {",
  "      return false;", # prevent moving a child above or below the root
  "    }",               # and moving inside a child
  "  }",
  "  return true;", # allow everything else
  "}"
)

dnd <- list(
  is_draggable = JS(
    "function(node) {",
    "  return node[0].type === 'child';",
    "}"
  )
)

jstree(
  nodes,
```

```
  dragAndDrop = TRUE, dnd = dnd,
  types = types,
  checkCallback = checkCallback
)


# example illustrating the 'grid' option ####

library(jsTreeR)

nodes <- list(
  list(
    text = "Products",
    children = list(
      list(
        text = "Fruit",
        children = list(
          list(
            text = "Apple",
            data = list(
              price = 0.1,
              quantity = 20
            )
          ),
          list(
            text = "Banana",
            data = list(
              price = 0.2,
              quantity = 31
            )
          ),
          list(
            text = "Grapes",
            data = list(
              price = 1.99,
              quantity = 34
            )
          ),
          list(
            text = "Mango",
            data = list(
              price = 0.5,
              quantity = 8
            )
          ),
          list(
            text = "Melon",
            data = list(
              price = 0.8,
              quantity = 4
            )
          ),
          list(
```

```
      text = "Pear",
      data = list(
        price = 0.1,
        quantity = 30
      )
    ),
    list(
      text = "Strawberry",
      data = list(
        price = 0.15,
        quantity = 32
      )
    )
  ),
  state = list(
    opened = TRUE
  )
),
list(
  text = "Vegetables",
  children = list(
    list(
      text = "Aubergine",
      data = list(
        price = 0.5,
        quantity = 8
      )
    ),
    list(
      text = "Broccoli",
      data = list(
        price = 0.4,
        quantity = 22
      )
    ),
    list(
      text = "Carrot",
      data = list(
        price = 0.1,
        quantity = 32
      )
    ),
    list(
      text = "Cauliflower",
      data = list(
        price = 0.45,
        quantity = 18
      )
    ),
    list(
      text = "Potato",
      data = list(
        price = 0.2,
```

```
                quantity = 38
              )
            )
          )
        )
      ),
      state = list(
        opened = TRUE
      )
    )
  )

  grid <- list(
    columns = list(
      list(
        width = 200,
        header = "Name"
      ),
      list(
        width = 150,
        value = "price",
        header = "Price"
      ),
      list(
        width = 150,
        value = "quantity",
        header = "Qty"
      )
    ),
    width = 600
  )

  jstree(nodes, grid = grid)
```

`jstree-shiny` *Shiny bindings for jstree*

Description

Output and render functions for using jstree within Shiny applications and interactive Rmd documents.

Usage

```
jstreeOutput(outputId, width = "100%", height = "auto")

renderJstree(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

outputId	output variable to read from
width, height	must be a valid CSS unit (like '100%', '400px', 'auto') or a number, which will be coerced to a string and have 'px' appended
expr	an expression that generates a jstree
env	the environment in which to evaluate expr
quoted	logical, whether expr is a quoted expression (with quote()); this is useful if you want to save an expression in a variable

Examples

```
# displaying a folder ####

library(jsTreeR)
library(shiny)
library(jsonlite)

# make the nodes list from a vector of file paths
makeNodes <- function(leaves){
  dfs <- lapply(strsplit(leaves, "/"), function(s){
    item <-
      Reduce(function(a,b) paste0(a,"/",b), s[-1], s[1], accumulate = TRUE)
    data.frame(
      item = item,
      parent = c("root", item[-length(item)]),
      stringsAsFactors = FALSE
    )
  })
  dat <- dfs[[1]]
  for(i in 2:length(dfs)){
    dat <- merge(dat, dfs[[i]], all = TRUE)
  }
  f <- function(parent){
    i <- match(parent, dat$item)
    item <- dat$item[i]
    children <- dat$item[dat$parent==item]
    label <- tail(strsplit(item, "/")[[1]], 1)
    if(length(children)){
      list(
        text = label,
        data = list(value = item),
        children = lapply(children, f)
      )
    }else{
      list(text = label, data = list(value = item))
    }
  }
  lapply(dat$item[dat$parent == "root"], f)
}
```

```
folder <-
  list.files(system.file("www", "shared", package = "shiny"), recursive = TRUE)
nodes <- makeNodes(folder)

ui <- fluidPage(
  br(),
  fluidRow(
    column(
      width = 4,
      jstreeOutput("jstree")
    ),
    column(
      width = 4,
      tags$fieldset(
        tags$legend("Selections - JSON format"),
        verbatimTextOutput("treeSelected_json")
      )
    ),
    column(
      width = 4,
      tags$fieldset(
        tags$legend("Selections - R list"),
        verbatimTextOutput("treeSelected_R")
      )
    )
  )
)

server <- function(input, output){

  output[["jstree"]] <-
    renderJstree(
      jstree(nodes, search = TRUE, checkboxes = TRUE)
    )

  output[["treeSelected_json"]] <- renderPrint({
    toJSON(input[["jstree_selected"]], pretty = TRUE, auto_unbox = TRUE)
  })

  output[["treeSelected_R"]] <- renderPrint({
    input[["jstree_selected"]]
  })

}

if(interactive()){
  shinyApp(ui, server)
}


# drag-and-drop, checkboxes, proton theme, fontawesome icons ####

library(jsTreeR)
```

```
library(shiny)
library(jsonlite)

nodes <- list(
  list(
    text = "RootA",
    data = list(value = 999),
    icon = "far fa-moon red",
    children = list(
      list(
        text = "ChildA1",
        icon = "fa fa-leaf green"
      ),
      list(
        text = "ChildA2",
        icon = "fa fa-leaf green"
      )
    )
  ),
  list(
    text = "RootB",
    icon = "far fa-moon red",
    children = list(
      list(
        text = "ChildB1",
        icon = "fa fa-leaf green"
      ),
      list(
        text = "ChildB2",
        icon = "fa fa-leaf green"
      )
    )
  )
)

ui <- fluidPage(

  tags$head(
    tags$style(
      HTML(c(
        ".red {color: red;}",
        ".green {color: green;}",
        ".jstree-proton {font-weight: bold;}",
        ".jstree-anchor {font-size: medium;}"
      ))
    )
  ),

  titlePanel("Drag and drop the nodes"),

  fluidRow(
    column(
      width = 4,
```

```
      jstreeOutput("jstree")
    ),
    column(
      width = 4,
      tags$fieldset(
        tags$legend("All nodes"),
        verbatimTextOutput("treeState")
      )
    ),
    column(
      width = 4,
      tags$fieldset(
        tags$legend("Selected nodes"),
        verbatimTextOutput("treeSelected")
      )
    )
  )

)

server <- function(input, output){

  output[["jstree"]] <- renderJstree({
    jstree(nodes, dragAndDrop = TRUE, checkboxes = TRUE, theme = "proton")
  })

  output[["treeState"]] <- renderPrint({
    toJSON(input[["jstree"]], pretty = TRUE, auto_unbox = TRUE)
  })

  output[["treeSelected"]] <- renderPrint({
    toJSON(input[["jstree_selected"]], pretty = TRUE, auto_unbox = TRUE)
  })

}

if(interactive()){
  shinyApp(ui, server)
}


# Super tiny icons, with 'search' options ####

library(jsTreeR)
library(shiny)
library(jsonlite)

nodes <- fromJSON(
  system.file(
    "htmlwidgets",
    "SuperTinyIcons",
    "SuperTinyIcons.json",
    package = "jsTreeR"
```

```
  ),
  simplifyDataFrame = FALSE
)

ui <- fluidPage(
  tags$head(
    tags$style(
      HTML(
        "#jstree {background-color: #fff5ee;}",
        "img {background-color: #333; padding: 50px;}"
      )
    )
  ),
  titlePanel("Super tiny icons"),
  fluidRow(
    column(
      width = 6,
      jstreeOutput("jstree", height = "auto")
    ),
    column(
      width = 6,
      checkboxInput("transparent", "Transparent background"),
      uiOutput("icon")
    )
  )
)

server <- function(input, output){
  output[["jstree"]] <- renderJstree({
    jstree(nodes, multiple = FALSE, search = list(
      show_only_matches = TRUE,
      case_sensitive = TRUE,
      search_leaves_only = TRUE
    ))
  })
  output[["icon"]] <- renderUI({
    req(length(input[["jstree_selected"]]) > 0)
    svg <- req(input[["jstree_selected"]][[1]][["data"]][["svg"]])
    if(input[["transparent"]])
      svg <- paste0("transparent-", svg)
    tags$img(src = paste0("/SuperTinyIcons/", svg), width = "75%")
  })
}

if(interactive()){
  shinyApp(ui, server)
}


# grid example ####

library(jsTreeR)
library(shiny)
```

```
nodes <- list(
  list(
    text = "Fruits",
    type = "fruit",
    icon = "supertinyicon-transparent-raspberry_pi",
    a_attr = list(class = "helvetica"),
    children = list(
      list(
        text = "Apple",
        type = "fruit",
        data = list(
          price = 0.1,
          quantity = 20,
          cssclass = "lightorange"
        )
      ),
      list(
        text = "Banana",
        type = "fruit",
        data = list(
          price = 0.2,
          quantity = 31,
          cssclass = "lightorange"
        )
      ),
      list(
        text = "Grapes",
        type = "fruit",
        data = list(
          price = 1.99,
          quantity = 34,
          cssclass = "lightorange"
        )
      ),
      list(
        text = "Mango",
        type = "fruit",
        data = list(
          price = 0.5,
          quantity = 8,
          cssclass = "lightorange"
        )
      ),
      list(
        text = "Melon",
        type = "fruit",
        data = list(
          price = 0.8,
          quantity = 4,
          cssclass = "lightorange"
        )
      ),
```

```
      list(
        text = "Pear",
        type = "fruit",
        data = list(
          price = 0.1,
          quantity = 30,
          cssclass = "lightorange"
        )
      ),
      list(
        text = "Strawberry",
        type = "fruit",
        data = list(
          price = 0.15,
          quantity = 32,
          cssclass = "lightorange"
        )
      )
    ),
    state = list(
      opened = TRUE
    )
  ),
  list(
    text = "Vegetables",
    type = "vegetable",
    icon = "supertinyicon-transparent-vegetarian",
    a_attr = list(class = "helvetica"),
    children = list(
      list(
        text = "Aubergine",
        type = "vegetable",
        data = list(
          price = 0.5,
          quantity = 8,
          cssclass = "lightgreen"
        )
      ),
      list(
        text = "Broccoli",
        type = "vegetable",
        data = list(
          price = 0.4,
          quantity = 22,
          cssclass = "lightgreen"
        )
      ),
      list(
        text = "Carrot",
        type = "vegetable",
        data = list(
          price = 0.1,
          quantity = 32,
```

```
            cssclass = "lightgreen"
          )
        ),
        list(
          text = "Cauliflower",
          type = "vegetable",
          data = list(
            price = 0.45,
            quantity = 18,
            cssclass = "lightgreen"
          )
        ),
        list(
          text = "Potato",
          type = "vegetable",
          data = list(
            price = 0.2,
            quantity = 38,
            cssclass = "lightgreen"
          )
        )
      )
    )
  )

  grid <- list(
    columns = list(
      list(
        width = 200,
        header = "Product",
        headerClass = "bolditalic yellow centered",
        wideValueClass = "cssclass"
      ),
      list(
        width = 150,
        value = "price",
        header = "Price",
        wideValueClass = "cssclass",
        headerClass = "bolditalic yellow centered",
        wideCellClass = "centered"
      ),
      list(
        width = 150,
        value = "quantity",
        header = "Quantity",
        wideValueClass = "cssclass",
        headerClass = "bolditalic yellow centered",
        wideCellClass = "centered"
      )
    ),
    width = 600
  )
```

```
types <- list(
  fruit = list(
    a_attr = list(
      class = "lightorange"
    ),
    icon = "supertinyicon-transparent-symantec"
  ),
  vegetable = list(
    a_attr = list(
      class = "lightgreen"
    ),
    icon = "supertinyicon-transparent-symantec"
  )
)

ui <- fluidPage(
  tags$head(
    tags$style(
      HTML(c(
        ".lightorange {background-color: #fed8b1;}",
        ".lightgreen {background-color: #98ff98;}",
        ".bolditalic {font-weight: bold; font-style: italic; font-size: large;}",
        ".yellow {background-color: yellow !important;}",
        ".centered {text-align: center; font-family: cursive;}",
        ".helvetica {font-weight: 700; font-family: Helvetica; font-size: larger;}"
      ))
    )
  ),
  titlePanel("jsTree grid"),
  jstreeOutput("jstree")
)

server <- function(input, output){
  output[["jstree"]] <-
    renderJstree(jstree(nodes, grid = grid, types = types))
}

if(interactive()){
  shinyApp(ui, server)
}


# Filtering ####

library(jsTreeR)
library(shiny)
library(htmlwidgets)
library(magrittr)

data("Countries")
rownames(Countries) <- Countries[["countryName"]]
dat <- split(Countries, Countries[["continentName"]])
nodes <- lapply(names(dat), function(continent){
```

```
    list(
      text = continent,
      children = lapply(dat[[continent]][["countryName"]], function(cntry){
        list(
          text = cntry,
          data = list(population = Countries[cntry, "population"])
        )
      })
    )
  })

  onrender <- c(
    "function(el, x) {",
    "  Shiny.addCustomMessageHandler('hideNodes', function(range) {",
    "    var tree = $.jstree.reference(el.id);",
    "    var json = tree.get_json(null, {flat: true});",
    "    for(var i = 0; i < json.length; i++) {",
    "      var id = json[i].id;",
    "      if(tree.is_leaf(id)) {",
    "        var pop = json[i].data.population;",
    "        if(pop < range[0] || pop > range[1]) {",
    "          tree.hide_node(id);",
    "        } else {",
    "          tree.show_node(id);",
    "        }",
    "      }",
    "    }",
    "  });",
    "}"
  )

  ui <- fluidPage(
    tags$h3("Open a node and filter with the slider."),
    br(),
    fluidRow(
      column(
        6,
        jstreeOutput("tree")
      ),
      column(
        6,
        sliderInput(
          "range",
          label = "Population",
          min = 0, max = 100000000, value = c(0, 100000000)
        )
      )
    )
  )

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

    output[["tree"]] <- renderJstree({
```

```
    jstree(nodes, checkboxes = TRUE) %>% onRender(onrender)
  })

  observeEvent(input[["range"]], {
    session$sendCustomMessage("hideNodes", input[["range"]])
  })

}

if(interactive()){
  shinyApp(ui, server)
}
```

`jsTreeR-imports` *Objects imported from other packages*

Description

These objects are imported from other packages. Follow the links to their documentation: JS, saveWidget.

Index