Package: shinyStorePlus (via r-universe)

December 6, 2024

Type Package

Title Secure in-Browser and Database Storage for 'shiny' Inputs, Outputs, Views and User Likes

Version 1.3

Maintainer Obinna Obianom <idonshayo@gmail.com>

Description Store persistent and synchronized data from 'shiny' inputs within the browser. Refresh 'shiny' applications and preserve user-inputs over multiple sessions. A database-like storage format is implemented using 'Dexie.js' <https://dexie.org>, a minimal wrapper for 'IndexedDB'. Transfer browser link parameters to 'shiny' input or output values. Store app visitor views, likes and followers.

License MIT + file LICENSE

URL https://shinystoreplus.obi.obianom.com

BugReports https://github.com/oobianom/shinyStorePlus/issues

Depends R (> 3.6)

Imports shiny, jsonlite, utils, htmltools, shinyWidgets

Suggests rmarkdown, knitr, qpdf

Encoding UTF-8

VignetteBuilder knitr

Language en-US

LazyData false

RoxygenNote 7.2.3

NeedsCompilation no

Author Obinna Obianom [aut, cre]

Repository CRAN

Date/Publication 2024-12-06 10:30:02 UTC

Config/pak/sysreqs make zlib1g-dev

Contents

`clearStore` *Clear storage for an application*

Description

Remove all stored inputs of an application

Usage

```
clearStore(appId, session = getDefaultReactiveDomain())
```

Arguments

`appId`	the application identification
`session`	session object

Value

No return value, called for side effects

Note

Ensure not to use this function when the inputs are intended to be tracked.

Examples

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

  ui <- fluidPage(
    # initialize stores
    initStore(),
    "Sample delete storage",
    selectInput("dataset",
      label = "Dataset",
      choices = c("dataset 1", "dataset 2")
    )
  )
```

```
  server <- function(input, output, session) {
    appid <- "application01"
    clearStore(appId = appid)
  }
  shinyApp(ui, server)
}
```

`initStore` *Included package scripts*

Description

To unlock the "rpkg" or "all" functionality, you'll need to obtain a FREE API key from https://api.rpkg.net However, before requesting your API key, it's recommended to do an initial deployment of your app. This is because the API key generation process requires you to provide the link to your Shiny app.

Usage

```
initStore(src = c("browser", "rpkg", "all"), rpkg.api.key)
```

Arguments

`src`	scripts to include
`rpkg.api.key`	API key obtained from rpkg.net to use if src = "rpkg" or "all"

Details

Include Dexie and the package script in the header

Value

Initialize the storage by including scripts necessary for the persistent storage handling

Note

Choices for "src":

"browser" - include only scripts relevant for storing data within browser

"rpkg" - include scripts relevant for storing app views, likes and followers in rpkg.net

"all" - include all scripts

Examples

```
library(shiny)
library(shinyStorePlus)

if (interactive()) {
  ui <- shiny::fluidPage(
    # initialize stores
    initStore(),
    titlePanel("Sample
             shinyStorePlus Init Inputs"),
    sidebarLayout(
      sidebarPanel(
        sliderInput("nextgenshinyapps1",
          "Number of bins:",
          min = 1,
          max = 200,
          value = 150
        ),
        textInput(
          "caption",
          "simple caption:",
          "summary, try editing"
        ),
        numericInput("obs",
          "sample observations:",
          10,
          min = 1, max = 100
        )
      ),
      mainPanel(
        plotOutput("distPlot")
      )
    )
  )
  server <- function(input, output, session) {
    output$distPlot <- renderPlot({
      x <- faithful[, 2]
      bins <- seq(min(x),
        max(x),
        length.out =
          input$nextgenshinyapps1 + 1
      )
      hist(x,
        breaks = bins,
        col = "blue",
        border = "gray"
      )
    })
  }
  shiny::shinyApp(ui = ui, server = server)
}
```

```
# Example app that stores App user views, likes and followers

library(shiny)
library(shinyStorePlus)

if (interactive()) {
ui <- fluidPage(
  titlePanel("Simplified shiny app storage of views, likes and followers"),
  initStore("all",rpkg.api.key =
  "c20c5eead7714c119dd3f20bd249a388e72db2aa0f9305d0380b683a37c5296a"),
  h2("Save App Views"),hr(),
  viewsBox("viewsshow","loading views..."),
  h2("Save App Likes, and allow user to Like!"),hr(),
  lfButton("liket",suffix="likes"),
  h2("Save App Followers, and allow user to Follow!"),hr(),
  lfButton("followt",suffix="followers"),
  hr(),p(p("Like or Follow and Refresh the page -
  the values are saved and the views are incremented."))
)

server <- function(input, output, session) {
  # set up views, likes and follows, leave as
  # NULL if you don't need tracking for either
  # in this case, we leave followID as NULL
  # since we don't need to use that
  setupRPKG(
    viewsID = "viewsshow",
    likesID = "liket",
    followID = "followt"
 )
}


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

---

link2input *Convert Browser Location Parameters to Shiny Input and Output Values*

Description

Parse the browser link and retrieve parameters for inclusion as values in inputs or outputs

Usage

```
link2input(..., inputtype = "default", session = getDefaultReactiveDomain())
```

Arguments

...	List of Shiny input IDs to match with window location parameters
inputtype	Type of inputs being included
session	Shiny session object

Value

Setting of the Shiny inputs to the values of the parameters in the browser link

Note

a great example of how to use this functionality can be found in https://cran.r-project.org/web/packages/shinyStorePlus/vignet

Examples

```
if (interactive()) {
  # within the server function
  server <- function(input, output, session) {
    link2input(
      cd323 = "name",
      datasetbin = "data",
      numberid = "num"
    )

    link2input(
      outputid = "outt",
      inputtype = "output"
    )
  }
}
```

---

seeexample *Load the example for the package*

Description

Example of a shiny application with secure in-browser storage of inputs when changed

Usage

```
seeexample(
  name = c("storeInputs", "browserLinkToInput", "shinyWidgetsExamples")
)
```

Arguments

name	the name of example to view. choices include storeInputs or browserLinkToInput or shinyWidgetsExamples

Value

An example of inputs persistently stored when changed and the page refreshed

Note

Changes made to the input or putputs will be saved and returned when the page is refresh within the same browser over different sessions. More examples are located at https://github.com/oobianom/aagarw30_shinyapps_to-shinyStorePlus

Examples

```
if (interactive()) {
  seeexample()
  seeexample("browserLinkToInput")
}
```

---

setupRPKG *Setup configuration for shiny page views, likes and followers*

Description

To unlock this functionality, you'll need to obtain a FREE API key from https://api.rpkg.net
However, before requesting your API key, it's recommended to do an initial deployment of your app. This is because the API key generation process requires you to provide the link to your Shiny app.

Usage

```
setupRPKG(
  viewsID = NULL,
  likesID = NULL,
  followID = NULL,
  session = getDefaultReactiveDomain(),
  icon.follow = shiny::icon("user"),
  icon.unfollow = shiny::icon("user", class = "fa-solid"),
  icon.like = shiny::icon("heart"),
  icon.unlike = shiny::icon("heart", class = "fa-solid"),
  text.follow = "",
  text.unfollow = "",
  text.like = "",
  text.unlike = ""
)

viewsBox(inputId, ...)

lfButton(inputId, width = NULL, suffix = "", ...)
```

Arguments

viewsID	Optional. The container ID to display views
likesID	Optional. The button ID to display likes
followID	Optional. The button ID to display followers
session	Optional. Current session to track
icon.follow	Optional. shiny::icon() to activate follow
icon.unfollow	Optional. shiny::icon() to de-activate follow
icon.like	Optional. shiny::icon() to activate likes
icon.unlike	Optional. shiny::icon() to de-activate likes
text.follow	Optional. text to activate follow
text.unfollow	Optional. text to de-activate follow
text.like	Optional. text to activate likes
text.unlike	Optional. text to de-activate likes
inputId	The input slot that will be used to access the value.
...	Optional. Named attributes to be applied to the likes or follows button or views box.
width	Optional. The width of the button input, e.g. '500px', or '100%'
suffix	suffix to add to likes or followers count
icon	Optional. A shiny::icon() to appear on the button.

Details

Utilize the rpkg.net API to store and retrieve page views, likes and followers

Examples

```
library(shiny)
library(shinyStorePlus)

if (interactive()) {

# replace UI with more elements
ui <- fluidPage(
  initStore("rpkg",rpkg.api.key =
  "c20c5eead7714c119dd3f20bd249a388e72db2aa0f9305d0380b683a37c5296a")
)

# this example is focused on the server
server <- function(input, output, session) {
  setupRPKG(
    session = session,
    viewsID = "viewsshow",
    likesID = "liket",
    followID = "followt",
    icon.follow = shiny::icon("user-plus"),
```

```
    icon.unfollow = shiny::icon("user-minus"),
    icon.like = shiny::icon("thumbs-up"),
    icon.unlike = shiny::icon("thumbs-down"),
    text.follow = "Follow us!",
    text.unfollow = "Unfollow us!",
    text.like = "Like us!",
    text.unlike = "Unlike us!"
 )
}


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

`setupStorage` *Set up inputs for storage*

Description

Set up the application and inputs to track and retrieve stores

Usage

```
setupStorage(
  appId,
  inputs = TRUE,
  outputs = FALSE,
  session = getDefaultReactiveDomain(),
  dyn.inputs = list()
)
```

Arguments

`appId`	your desired application id
`inputs`	choose whether to track all inputs or specific input variables
`outputs`	choose whether to track all outputs or specific output variables
`session`	current session to track
`dyn.inputs`	dynamic inputs; inputs that get added to the app from the server function

Details

As of version 1.2, the user may be able to store dynamically generated inputs

Value

Embed within a page storage that allows input changes to be saved without slowing down the shiny application

Note

the inputs argument may be a TRUE or FALSE or a list of input ids. More examples are located at https://github.com/oobianom/aagarw30_shinyapps_to-shinyStorePlus

Examples

```
library(shiny)
library(shinyStorePlus)

# example 1 that tracks all inputs
if (interactive()) {
  ui <- shiny::fluidPage(
    titlePanel("EX1
             shinyStorePlus All Inputs"),
    initStore(),
    sidebarLayout(
      sidebarPanel(
        sliderInput("nextgenshinyapps1",
          "Number of bins:",
          min = 1,
          max = 200,
          value = 150
        ),
        textInput(
          "caption",
          "simple caption:",
          "try editing - r2resize pkg"
        ),
        numericInput("obs",
          "sample observations:",
          10,
          min = 1, max = 100
        )
      ),
      mainPanel(
        plotOutput("distPlot")
      )
    )
  )
  server <- function(input, output, session) {
    output$distPlot <- renderPlot({
      x <- faithful[, 2]
      bins <- seq(min(x),
        max(x),
        length.out =
          input$nextgenshinyapps1 + 1
      )
```

```
      hist(x,
        breaks = bins,
        col = "blue",
        border = "gray"
      )
    })

    # insert at the bottom
    appid <- "application01"
    setupStorage(
      appId = appid,
      inputs = TRUE
    )
  }
  shiny::shinyApp(ui = ui, server = server)
}


# example 2 that tracks only 2 inputs
if (interactive()) {
  ui <- shiny::fluidPage(
    # init stores
    initStore(),
    titlePanel("Ex2:
             shinyStorePlus Some Inputs"),
    sidebarLayout(
      sidebarPanel(
        sliderInput("nextgenshinyapps1",
          "Number of bins:",
          min = 1,
          max = 200,
          value = 150
        ),
        textInput(
          "caption",
          "simple caption:",
          "summary, try editing"
        ),
        numericInput("obs",
          "sample observations:",
          10,
          min = 1, max = 100
        )
      ),
      mainPanel(
        plotOutput("distPlot")
      )
    )
  )
  server <- function(input, output, session) {
    output$distPlot <- renderPlot({
      x <- faithful[, 2]
      bins <- seq(min(x),
```

```
      max(x),
      length.out =
        input$nextgenshinyapps1 + 1
    )
    hist(x,
      breaks = bins,
      col = "blue",
      border = "gray"
    )
  })

  # insert at the bottom  !!!IMPORTANT
  appid <- "application023"
  setupStorage(
    appId = appid,
    inputs = list(
      "nextgenshinyapps1",
      "caption"
    )
  )
}
shiny::shinyApp(ui = ui, server = server)
}

# example 3 with dynamically generated inputs
if(interactive()){
  ui <- shiny::fluidPage(
    titlePanel("Select option,
               then referesh page."),
    initStore(),
    selectInput("sel_color",
                "Color (hardcoded input):",
                choices = c("", "green", "blue",
                            "red", "yellow",
                            "cyan"), selected = ""),
    uiOutput("ui_moreinputs")
  )

  server <- function(input, output, session) {
    observe({
      output$ui_moreinputs <- renderUI(
        selectInput("sel_month",
                    "Month (dynamically generated):",
                    choices = c("", month.name),
                    selected = "")
      )
    })

    setupStorage(appId = "dynamic02",
                 inputs = list("sel_color"),
                 dyn.inputs = list("sel_month"),
                 session = session)
  }
```

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

Index