

Package ‘shinyStorePlus’

July 23, 2025

Type Package

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

Version 1.6

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 2025-06-06 08:20:02 UTC

Contents

clearStore	2
initStore	3
link2input	5
observeOnce	6
seeexample	8
setupRPKG	9
setupStorage	11
Index	15

clearStore	<i>Clear storage for an application</i>
------------	---

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

Include Dexie and the package script in the header

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

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.

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"),
    # get API key from https://api.rpkg.net
    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

<code>...</code>	List of Shiny input IDs to match with window location parameters
<code>inputtype</code>	Type of inputs being included
<code>session</code>	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/vignettes/>

Examples

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

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

observeOnce

Observe event execution ONCE across multiple sessions

Description

Observe event that only gets executed one across multiple shiny sessions

Usage

```
observeOnce(
  expression,
  session = getDefaultReactiveDomain(),
  input,
  output,
  priority = 1
)

observeOnceRestart(session = getDefaultReactiveDomain())
```

Arguments

expression	An expression (quoted or unquoted). Any return value will be ignored.
session	the shiny server session object passed to function. Default is getDefaultReactiveDomain()
input	the shiny server input
output	the shiny server output
priority	An integer or numeric that controls the priority with which this observer should be executed. A higher value means higher priority: an observer with a higher priority value will execute before all observers with lower priority values. Positive, negative, and zero values are allowed.

Details

The current set of functions enables users to execute an expression in the 'Shiny' app ONCE across multiple sessions, ensuring that certain actions or elements only appear the first time a user interacts with the app. For example, it could be used to display a welcome modal or a cookie acceptance prompt only during the user's first session. After the initial interaction, these elements will not appear again, even if the user refreshes the app or closes and reopens it later. This functionality helps streamline the user experience by preventing repetitive prompts and maintaining a clean interface for returning users.

Examples

```
library(shiny)
library(shinyStorePlus)

if (interactive()) {
  ui <- fluidPage(
    titlePanel("A shiny app that shows welcome message only
once accross multiple sessions. Refresh and
see it in action"),
    initStore("browser"), sidebarPanel(
      sliderInput("slidex", label = "Sample slider",
        min = 1, max = 50, value = 30)
    ),
    shiny::mainPanel(width=6, tags$h1(textOutput("slidexresres")),
      "This functionality helps streamline the user
experience by preventing repetitive prompts
and maintaining a clean interface for returning users.")
  )

  server <- function(input, output, session) {
    # the observe expression below will only
    # execute once in 30 days, regardless of
    # if the user opens or refresh the shiny app many times
    observeOnce({
      output$slidexresres <- renderText(input$slidex)
      showModal(modalDialog(
```

```

    title = "Welcome to my app",
    "Welcome to the app! We're excited to have you here.
    Please note that this message will only appear the first time you visit.
    After this, it won't show up again, even if you refresh or reopen the app.
    Feel free to explore, and we hope you enjoy your experience!",
    size = "m"
  ))
}, input = input, output = output)

# optional: clear the storage for the previous time it was executed
#observeOnceRestart()
}

shinyApp(ui = ui, server = server)
}

```

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 outputs 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

<code>icon.unfollow</code>	Optional. <code>shiny::icon()</code> to de-activate follow
<code>icon.like</code>	Optional. <code>shiny::icon()</code> to activate likes
<code>icon.unlike</code>	Optional. <code>shiny::icon()</code> to de-activate likes
<code>text.follow</code>	Optional. text to activate follow
<code>text.unfollow</code>	Optional. text to de-activate follow
<code>text.like</code>	Optional. text to activate likes
<code>text.unlike</code>	Optional. text to de-activate likes
<code>inputId</code>	The input slot that will be used to access the value.
<code>...</code>	Optional. Named attributes to be applied to the likes or follows button or views box.
<code>width</code>	Optional. The width of the button input, e.g. <code>'500px'</code> , or <code>'100%'</code>
<code>suffix</code>	suffix to add to likes or followers count
<code>icon</code>	Optional. A <code>shiny::icon()</code> 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 =
      "c20c5ead7714c119dd3f20bd249a388e72db2aa0f9305d0380b683a37c5296a")
  )

  # 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	the session object passed to function. Default is getDefaultReactiveDomain()
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

`clearStore`, [2](#)

`initStore`, [3](#)

`lfButton (setupRPKG)`, [9](#)

`link2input`, [5](#)

`observeOnce`, [6](#)

`observeOnceRestart (observeOnce)`, [6](#)

`seeexample`, [8](#)

`setupRPKG`, [9](#)

`setupStorage`, [11](#)

`viewsBox (setupRPKG)`, [9](#)