Package: shinystate (via r-universe)

October 21, 2024

Title Customization of Shiny Bookmarkable State

Version 0.0.0.9001

Description Enhance the bookmarkable state feature of Shiny with additional customization such as storage location and storage repositories leveraging the pins package.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports archive, dplyr, fs, htmltools, pins, R6, shiny (>= 0.14), tibble

Suggests DT, lubridate, rlang, shinytest2, testthat (>= 3.0.0), withr

Config/testthat/edition 3

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

RemoteUrl https://github.com/rpodcast/shinystate

RemoteRef HEAD

RemoteSha 4c71bcae8d4bf0f87bfb99cc62f93fcc49e07f79

Contents

StorageClass 2
use_shinystate 9

Index **10**

`StorageClass` *StorageClass R6 class*

Description

This class provides a set of methods to create and manage Shiny bookmarkable state files.

Public fields

`local_storage_dir` file path to use for storing bookmarkable state files. If not specified, a temporary directory on the host system will be used.

`board_sessions` Optional pre-created board object created with the pins package. If missing, a folder-based pin board will be created using the `local_storage_dir` path.

Methods

Public methods:

- `StorageClass$new()`
- `StorageClass$get_sessions()`
- `StorageClass$restore()`
- `StorageClass$snapshot()`
- `StorageClass$delete()`
- `StorageClass$register_metadata()`
- `StorageClass$clone()`

Method `new()`**:** Initialize a `StorageClass` object

Usage:

```
StorageClass$new(local_storage_dir = NULL, board_sessions = NULL)
```

Arguments:

`local_storage_dir` file path to use for storing bookmarkable state files. If not specified, a temporary directory on the host system will be used.

`board_sessions` Optional pre-created board object created with the pins package. If missing, a folder-based pin board will be created using the `local_storage_dir` path.

Returns: An object with class `StorageClass` and the methods described in this documentation

Examples:

```
\dontrun{
# beginning of application
library(shiny)
library(shinystate)

# Create a StorageClass object with default settings
storage <- StorageClass$new()
# Use a local directory called "sessions" to store files
storage <- StorageClass$new(local_storage_dir = "sessions")

# use a custom pins board to store bookmarkable state data
library(pins)
board <- board_folder("/path/to/storage_dir")
storage <- StorageClass$new(board_sessions = board)
}
```

Method `get_sessions()`**:** Obtain saved bookmarkable state session metadata

Calls `$get_sessions()` on the `StorageClass` object to extract the bookmarkable state session metadata. You can leverage this data frame in your Shiny application to let the user manage their existing bookmarkable state sessions, for example.

Usage:

```
StorageClass$get_sessions()
```

Examples:

```
\dontrun{
# beginning of application
library(shiny)
library(shinystate)

storage <- StorageClass$new()

# application UI for displaying session data
DT::datatableOutput("session_table")

# server logic for displaying session data
output$session_table <- DT::renderDT({
  storage$get_sessions()
})
}
```

Method `restore()`**:** Restore a previous bookmarkable state session

Usage:

```
StorageClass$restore(url, session = shiny::getDefaultReactiveDomain())
```

Arguments:

`url` character with the unique URL assigned to the bookmarkable state session.

`session` The Shiny session to associate with the restore operation

Examples:

```
\dontrun{
# beginning of application
library(shiny)
library(shinystate)

# restoration of last-saved bookmarkable state file
#
# beginning of application
storage <- StorageClass$new()

# application UI to trigger restore
shiny::actionButton("restore", "Restore State")

# server logic for restoring state
observeEvent(input$restore, {
  session_df <- storage$get_sessions()
  storage$restore(tail(session_df$url, n = 1))
})
}
```

Method `snapshot()`**:** Create a snapshot of bookmarkable state

Usage:

```
StorageClass$snapshot(
  session_metadata = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments:

`session_metadata` Optional named list of additional variables to include with the default bookmarkable state attributes when creating the snapshot. Each element of the list must be a single-length item

`session` The Shiny session to associate with the snapshot operation

Examples:

```
\dontrun{
# beginning of application
library(shiny)
library(shinystate)

storage <- StorageClass$new()

# application UI to trigger save
actionButton("save", "Save State")

# server logic for restoring state with timestamp as metadata
observeEvent(input$save, {
  storage$snapshot(session_metadata = list(time = Sys.time()))
})
}
```

Method `delete()`**:** Delete a previous snapshot of bookmarkable state

Usage:

```
StorageClass$delete(url)
```

Arguments:

`url` character with the unique URL assigned to the bookmarkable state session.

Examples:

```
\dontrun{
# beginning of application
library(shiny)
library(shinystate)

storage <- StorageClass$new()

# application UI to let user choose previous session
uiOutput("previous_sessions_ui")

# application UI to trigger delete
shiny::actionButton("delete", "Delete Session")

# server logic
# populate dynamic UI
output$previous_sessions_ui <- renderUI({
  session_df <- storage$get_sessions
  radioButtons(
    "session_choice",
    "Choose Session",
    choices = session_df$url
  )
})

# perform session deletion
observeEvent(input$delete, {
  req(input$session_choice)
  storage$delete(input$session_choice)
})
}
```

Method `register_metadata()`**:** Register bookmarkable state storage data collection

This method must be called in the application server function to perform the necessary customizations to bookmark methods

Usage:

```
StorageClass$register_metadata()
```

Examples:

```
\dontrun{
# beginning of application
library(shiny)
library(shinystate)

storage <- StorageClass$new()

# applicaiton UI code ...
# application server code
storage$register_metadata()
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`StorageClass$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `StorageClass$new`
## ------------------------------------------------

## Not run:
# beginning of application
library(shiny)
library(shinystate)

# Create a StorageClass object with default settings
storage <- StorageClass$new()

# Use a local directory called "sessions" to store files
storage <- StorageClass$new(local_storage_dir = "sessions")

# use a custom pins board to store bookmarkable state data
library(pins)
board <- board_folder("/path/to/storage_dir")
storage <- StorageClass$new(board_sessions = board)

## End(Not run)

## ------------------------------------------------
## Method `StorageClass$get_sessions`
## ------------------------------------------------

## Not run:
# beginning of application
library(shiny)
library(shinystate)

storage <- StorageClass$new()

# application UI for displaying session data
DT::datatableOutput("session_table")

# server logic for displaying session data
output$session_table <- DT::renderDT({
  storage$get_sessions()
})

## End(Not run)

## ------------------------------------------------
## Method `StorageClass$restore`
## ------------------------------------------------

## Not run:
# beginning of application
library(shiny)
library(shinystate)

# restoration of last-saved bookmarkable state file
#
# beginning of application
storage <- StorageClass$new()

# application UI to trigger restore
shiny::actionButton("restore", "Restore State")

# server logic for restoring state
observeEvent(input$restore, {
  session_df <- storage$get_sessions()
  storage$restore(tail(session_df$url, n = 1))
})

## End(Not run)

## ------------------------------------------------
## Method `StorageClass$snapshot`
## ------------------------------------------------

## Not run:
# beginning of application
library(shiny)
library(shinystate)

storage <- StorageClass$new()

# application UI to trigger save
actionButton("save", "Save State")

# server logic for restoring state with timestamp as metadata
observeEvent(input$save, {
  storage$snapshot(session_metadata = list(time = Sys.time()))
})

## End(Not run)

## ------------------------------------------------
## Method `StorageClass$delete`
## ------------------------------------------------
## Not run:
# beginning of application
library(shiny)
library(shinystate)

storage <- StorageClass$new()

# application UI to let user choose previous session
uiOutput("previous_sessions_ui")

# application UI to trigger delete
shiny::actionButton("delete", "Delete Session")

# server logic
# populate dynamic UI
output$previous_sessions_ui <- renderUI({
  session_df <- storage$get_sessions
  radioButtons(
    "session_choice",
    "Choose Session",
    choices = session_df$url
  )
})

# perform session deletion
observeEvent(input$delete, {
  req(input$session_choice)
  storage$delete(input$session_choice)
})

## End(Not run)

## ------------------------------------------------
## Method `StorageClass$register_metadata`
## ------------------------------------------------

## Not run:
# beginning of application
library(shiny)
library(shinystate)

storage <- StorageClass$new()

# applicaiton UI code ...

# application server code
storage$register_metadata()

## End(Not run)
```

`use_shinystate` *Dependencies*

Description

Include shinystate dependencies in your Shiny UI

Usage

```
use_shinystate()
```

Index

StorageClass, 2, *3*

use_shinystate, 9