Package 'shinylogs'

August 21, 2019

Title Record Everything that Happens in a 'Shiny' Application

Version 0.1.7

Description Track and record the use of applications and the user's interactions with 'Shiny' inputs. Allow to save inputs clicked, output generated and eventually errors.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

URL https://github.com/dreamRs/shinylogs

BugReports https://github.com/dreamRs/shinylogs/issues

Imports htmltools, shiny (>= 1.1.0), jsonlite, data.table, bit64, nanotime, digest, anytime, DBI, RSQLite

Suggests testthat

NeedsCompilation no

Author Fanny Meyer [aut],
Victor Perrier [aut, cre],
Silex Technologies [fnd] (https://www.silex-ip.com),
iamkun [cph] (dayjs library),
Mozilla [cph] (localForage library)

Maintainer Victor Perrier <victor.perrier@dreamrs.fr>

Repository CRAN

Date/Publication 2019-08-21 17:10:02 UTC

R topics documented:

`read_json_logs` *Read a directory containing JSON logs*

Description

Read a directory containing JSON logs

Usage

```
read_json_logs(path)
```

Arguments

`path`	Path of the directory containing JSON files or a vector of path to JSON files.

Value

a list of `data.table`

Examples

```
# Read all JSON in a directory
path_directory <- system.file("extdata/json", package = "shinylogs")
logs <- read_json_logs(path = path_directory)

# Read a single file
single_file <- dir(
  path = system.file("extdata/json", package = "shinylogs"),
  full.names = TRUE
)[1]
logs <- read_json_logs(path = single_file)
```

read_rds_logs *Read a directory containing RDS logs*

Description

Read a directory containing RDS logs

Usage

```
read_rds_logs(path)
```

Arguments

path	Path of the directory containing RDS files or a vector of path to RDS files.

Value

a list of data.table

Examples

```
# Read all RDS in a directory
logs <- read_rds_logs(path = "path/to/directory")

# Read a single file
logs <- read_rds_logs(path = "path/to/log.rds")
```

store_json *Use JSON files as storage mode*

Description

One JSON will be written for each session of the application.

Usage

```
store_json(path)
```

Arguments

path	Path where to write JSON files.

Examples

```
if (interactive()) {

  # temp directory for writing logs
  tmp <- tempdir()

  # when app stop,
  # navigate to the directory containing logs
  onStop(function() {
    browseURL(url = tmp)
  })

  # Classir Iris clustering with Shiny
  ui <- fluidPage(

    headerPanel("Iris k-means clustering"),

    sidebarLayout(
      sidebarPanel(
        selectInput(
          inputId = "xcol",
          label = "X Variable",
          choices = names(iris)
        ),
        selectInput(
          inputId = "ycol",
          label = "Y Variable",
          choices = names(iris),
          selected = names(iris)[[2]]
        ),
        numericInput(
          inputId = "clusters",
          label = "Cluster count",
          value = 3,
          min = 1,
          max = 9
        )
      ),
      mainPanel(
        plotOutput("plot1")
      )
    )
  )

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

    # Store JSON with logs in the temp dir
    track_usage(
      storage_mode = store_json(path = tmp)
    )

    # classic server logic
```

```
    selectedData <- reactive({
      iris[, c(input$xcol, input$ycol)]
    })

    clusters <- reactive({
      kmeans(selectedData(), input$clusters)
    })

    output$plot1 <- renderPlot({
      palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
                "#FF7F00", "#FFFF33", "#A65628", "#F781BF", "#999999"))

      par(mar = c(5.1, 4.1, 0, 1))
      plot(selectedData(),
           col = clusters()$cluster,
           pch = 20, cex = 3)
      points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
    })

  }

  shinyApp(ui, server)

}
```

`store_null` *No storage on disk*

Description

Doesn't write anything, special inputs created by `track_usage` are available in server and optionally logs are printed in console.

Usage

```
store_null(console = TRUE)
```

Arguments

`console`	Print logs in R console.

Examples

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

  ui <- fluidPage(
    tags$h2("Record inputs change"),
```

```
    fluidRow(
      column(
        width = 3,
        selectInput(
          inputId = "select",
          label = "Select input",
          choices = month.name
        ),
        numericInput(
          inputId = "numeric",
          label = "Numerci input",
          value = 4,
          min = 0, max = 20
        ),
        checkboxGroupInput(
          inputId = "checkboxGroup",
          label = "Checkbox group input",
          choices = LETTERS[1:5]
        ),
        sliderInput(
          inputId = "slider",
          label = "Slider input",
          min = 0, max = 100, value = 50
        )
      ),
      column(
        width = 9,
        tags$b("Last input:"),
        verbatimTextOutput(outputId = "last_input"),
        tags$b("Last input:"),
        verbatimTextOutput(outputId = "all_inputs")
      )
    )
  )

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

    track_usage(
      storage_mode = store_null() # dont store on disk
    )

    output$last_input <- renderPrint({
      input$.shinylogs_lastInput # last input triggered
    })

    output$all_inputs <- renderPrint({
      input$.shinylogs_input # all inputs that have changed
    })

  }

  shinyApp(ui, server)
}
```

store_rds *Use RDS files as storage mode*

Description

One RDS will be written for each session of the application.

Usage

```
store_rds(path)
```

Arguments

path	Path where to write RDS files.

Examples

```
if (interactive()) {

  # temp directory for writing logs
  tmp <- tempdir()

  # when app stop,
  # navigate to the directory containing logs
  onStop(function() {
    browseURL(url = tmp)
  })

  # Classir Iris clustering with Shiny
  ui <- fluidPage(

    headerPanel("Iris k-means clustering"),

    sidebarLayout(
      sidebarPanel(
        selectInput(
          inputId = "xcol",
          label = "X Variable",
          choices = names(iris)
        ),
        selectInput(
          inputId = "ycol",
          label = "Y Variable",
          choices = names(iris),
          selected = names(iris)[[2]]
        ),
        numericInput(
          inputId = "clusters",
          label = "Cluster count",
          value = 3,
```

```
        min = 1,
        max = 9
      )
    ),
    mainPanel(
      plotOutput("plot1")
    )
  )
)

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

  # Store RDS with logs in the temp dir
  track_usage(
    storage_mode = store_rds(path = tmp)
  )

  # classic server logic

  selectedData <- reactive({
    iris[, c(input$xcol, input$ycol)]
  })

  clusters <- reactive({
    kmeans(selectedData(), input$clusters)
  })

  output$plot1 <- renderPlot({
    palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
              "#FF7F00", "#FFFF33", "#A65628", "#F781BF", "#999999"))

    par(mar = c(5.1, 4.1, 0, 1))
    plot(selectedData(),
         col = clusters()$cluster,
         pch = 20, cex = 3)
    points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
  })

}

shinyApp(ui, server)

}
```

`store_sqlite` *Use SQLite database as storage mode*

Description

All logs will be written in the same file.

Usage

```
store_sqlite(path)
```

Arguments

path	Path to the SQLite file or a directory where to create one.

Examples

```
if (interactive()) {

  # temp directory for writing logs
  tmp <- tempdir()

  # when app stop,
  # navigate to the directory containing logs
  onStop(function() {
    browseURL(url = tmp)
  })

  # Classir Iris clustering with Shiny
  ui <- fluidPage(

    headerPanel("Iris k-means clustering"),

    sidebarLayout(
      sidebarPanel(
        selectInput(
          inputId = "xcol",
          label = "X Variable",
          choices = names(iris)
        ),
        selectInput(
          inputId = "ycol",
          label = "Y Variable",
          choices = names(iris),
          selected = names(iris)[[2]]
        ),
        numericInput(
          inputId = "clusters",
          label = "Cluster count",
          value = 3,
          min = 1,
          max = 9
        )
      ),
      mainPanel(
        plotOutput("plot1")
      )
    )
  )
```

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

    # Store RDS with logs in the temp dir
    track_usage(
      storage_mode = store_sqlite(path = tmp)
    )

    # classic server logic

    selectedData <- reactive({
      iris[, c(input$xcol, input$ycol)]
    })

    clusters <- reactive({
      kmeans(selectedData(), input$clusters)
    })

    output$plot1 <- renderPlot({
      palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
                "#FF7F00", "#FFFF33", "#A65628", "#F781BF", "#999999"))

      par(mar = c(5.1, 4.1, 0, 1))
      plot(selectedData(),
           col = clusters()$cluster,
           pch = 20, cex = 3)
      points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
    })

  }

  shinyApp(ui, server)

}
```

`track_usage` *Track usage of a Shiny app*

Description

Used in Shiny `server` it will record all inputs and output changes and errors that occurs through an output.

Usage

```
track_usage(storage_mode, exclude_input_regex = NULL,
  exclude_input_id = NULL, on_unload = FALSE, exclude_users = NULL,
  get_user = NULL, dependencies = TRUE,
  session = getDefaultReactiveDomain())
```

Arguments

storage_mode	Storage mode to use : store_json, store_rds, store_sqlite or store_null.
exclude_input_regex	Regular expression to exclude inputs from tracking.
exclude_input_id	Vector of inputId to exclude from tracking.
on_unload	Logical, save log when user close the browser window or tab, if TRUE it prevent to create shinylogs input during normal use of the application, there will be created only on close, downside is that a popup will appear asking to close the page.
exclude_users	Character vectors of user for whom it is not necessary to save the log.
get_user	A function to get user name, it should return a character and take one argument: the Shiny session.
dependencies	Load dependencies.
session	The shiny session.

Note

The following inputs will be accessible in the server:

- **.shinylogs_lastInput** : last input used by the user
- **.shinylogs_input** : all inputs send from the browser to the server
- **.shinylogs_error** : all errors generated by outputs elements
- **.shinylogs_output** : all outputs generated from the server
- **.shinylogs_browserData** : information about the browser where application is displayed.

Examples

```
## Save logs on disk

if (interactive()) {

  # temporary directory for writing logs
  tmp <- tempdir()

  # when app stop,
  # navigate to the directory containing logs
  onStop(function() {
    browseURL(url = tmp)
  })

  # Classic Iris clustering with Shiny
  ui <- fluidPage(

    headerPanel("Iris k-means clustering"),
```

```
  sidebarLayout(
    sidebarPanel(
      selectInput(
        inputId = "xcol",
        label = "X Variable",
        choices = names(iris)
      ),
      selectInput(
        inputId = "ycol",
        label = "Y Variable",
        choices = names(iris),
        selected = names(iris)[[2]]
      ),
      numericInput(
        inputId = "clusters",
        label = "Cluster count",
        value = 3,
        min = 1,
        max = 9
      )
    ),
    mainPanel(
      plotOutput("plot1")
    )
  )
)

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

  # Store JSON with logs in the temp dir
  track_usage(
    storage_mode = store_json(path = tmp)
  )

  # classic server logic

  selectedData <- reactive({
    iris[, c(input$xcol, input$ycol)]
  })

  clusters <- reactive({
    kmeans(selectedData(), input$clusters)
  })

  output$plot1 <- renderPlot({
    palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
              "#FF7F00", "#FFFF33", "#A65628", "#F781BF", "#999999"))

    par(mar = c(5.1, 4.1, 0, 1))
    plot(selectedData(),
         col = clusters()$cluster,
         pch = 20, cex = 3)
    points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
```

```
    })

  }

  shinyApp(ui, server)

}


## Special inputs :

if (interactive()) {
  library(shiny)
  library(shinylogs)

  ui <- fluidPage(
    tags$h2("Record inputs change"),
    fluidRow(
      column(
        width = 3,
        selectInput(
          inputId = "select",
          label = "Select input",
          choices = month.name
        ),
        numericInput(
          inputId = "numeric",
          label = "Numerci input",
          value = 4,
          min = 0, max = 20
        ),
        checkboxGroupInput(
          inputId = "checkboxGroup",
          label = "Checkbox group input",
          choices = LETTERS[1:5]
        ),
        sliderInput(
          inputId = "slider",
          label = "Slider input",
          min = 0, max = 100, value = 50
        )
      ),
      column(
        width = 9,
        tags$b("Last input:"),
        verbatimTextOutput(outputId = "last_input"),
        tags$b("Last input:"),
        verbatimTextOutput(outputId = "all_inputs")
      )
    )
  )

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

```
    track_usage(
      storage_mode = store_null() # dont store on disk
    )

    output$last_input <- renderPrint({
      input$.shinylogs_lastInput # last input triggered
    })

    output$all_inputs <- renderPrint({
      input$.shinylogs_input # all inputs that have changed
    })

  }

  shinyApp(ui, server)
}
```

`use_tracking` *Insert dependencies to track usage of a Shiny app*

Description

If used in `ui` of an application, this will create new `inputs` available in the server. Set `dependencies = FALSE` in `track_usage` server-side to load dependencies only once.

Usage

```
use_tracking(on_unload = FALSE, exclude_input_regex = NULL,
  exclude_input_id = NULL)
```

Arguments

`on_unload`	Logical, save log when user close the browser window or tab, if `TRUE` it prevent to create `shinylogs` input during normal use of the application, there will be created only on close, downside is that a popup will appear asking to close the page.
`exclude_input_regex`	Regular expression to exclude inputs from tracking.
`exclude_input_id`	Vector of `inputId` to exclude from tracking.

Note

The following `inputs` will be accessible in the server:

- **.shinylogs_lastInput** : last `input` used by the user

- **.shinylogs_input** : all `inputs` send from the browser to the server

- **.shinylogs_error** : all errors generated by `outputs` elements
- **.shinylogs_output** : all `outputs` generated from the server
- **.shinylogs_browserData** : information about the browser where application is displayed.

Examples

```
if (interactive()) {

  library(shiny)

  ui <- fluidPage(

    use_tracking(),

    splitLayout(
      cellArgs = list(style = "height: 250px"),
      radioButtons("radio", "Radio:", names(iris)),
      checkboxGroupInput("checkbox", "Checkbox:", names(iris)),
      selectInput("select", "Select:", names(iris))
    ),

    verbatimTextOutput("last")
  )

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

    output$last <- renderPrint({
      input$.shinylogs_lastInput
    })

  }

  shinyApp(ui, server)

}
```

Index