

Package ‘shinymanager’

June 19, 2019

Title Authentication Management for 'Shiny' Applications

Version 1.0

Description Simple and secure authentication mechanism for single 'Shiny' applications. Credentials are stored in an encrypted 'SQLite' database. Source code of main application is protected until authentication is successful.

License GPL-3

URL <https://github.com/datastorm-open/shinymanager>

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Imports R6, shiny, htmltools, DT (>= 0.5), DBI, RSQLite, openssl, R.utils, billboardr

Suggests keyring, testthat (>= 2.1.0)

NeedsCompilation no

Author Benoit Thieurmel [aut, cre],
Victor Perrier [aut]

Maintainer Benoit Thieurmel <benoit.thieurmel@datastorm.fr>

Repository CRAN

Date/Publication 2019-06-19 11:20:04 UTC

R topics documented:

check_credentials	2
create_db	3
db-crypted	4
fab_button	6
generate_pwd	8
module-authentication	8
module-password	11
secure-app	12

Index	15
--------------	-----------

check_credentials *Check credentials*

Description

Check credentials

Usage

```
check_credentials(db, passphrase = NULL)
```

Arguments

db	A data.frame with credentials data or path to SQLite database created with create_db .
passphrase	Passphrase to decrypt the SQLite database.

Details

The credentials data.frame can have the following columns:

- **user (mandatory)** : the user's name.
- **password (mandatory)** : the user's password.
- **admin (optional)** : logical, is user have admin right ? If so, user can access the admin mode (only available using a SQLite database)
- **start (optional)** : the date from which the user will have access to the application
- **expire (optional)** : the date from which the user will no longer have access to the application
- **applications (optional)** : the name of the applications to which the user is authorized, separated by a semicolon. The name of the application corresponds to the name of the directory, or can be declared using : options("shinymanager.application" = "my-app")
- **additional columns** : add others columns to retrieve the values server-side after authentication

Value

Return a function with two arguments: user and password to be used in [module-authentication](#). The authentication function returns a list with 3 slots :

- **result** : logical, result of authentication.
- **expired** : logical, is user has expired ? Always FALSE if db doesn't have a expire column.
- **user_info** : the line in db corresponding to the user.

Examples

```
# data.frame with credentials info
credentials <- data.frame(
  user = c("fanny", "victor"),
  password = c("azerty", "12345"),
  stringsAsFactors = FALSE
)

# check a user
check_credentials(credentials)("fanny", "azerty")
check_credentials(credentials)("fanny", "azert")
check_credentials(credentials)("fannyyy", "azerty")

## Not run:

## With a SQLite database:

check_credentials("credentials.sqlite", passphrase = "supersecret")

## End(Not run)
```

create_db

Create credentials database

Description

Create a SQLite database with credentials data protected by a password.

Usage

```
create_db(credentials_data, sqlite_path, passphrase = NULL)
```

Arguments

credentials_data	A data.frame with information about users, user and password are required.
sqlite_path	Path to the SQLite database.
passphrase	A password to protect the data inside the database.

Details

The credentials data.frame can have the following columns:

- **user (mandatory)** : the user's name.
- **password (mandatory)** : the user's password.

- **admin (optional)** : logical, is user have admin right ? If so, user can access the admin mode (only available using a SQLite database)
- **start (optional)** : the date from which the user will have access to the application
- **expire (optional)** : the date from which the user will no longer have access to the application
- **applications (optional)** : the name of the applications to which the user is authorized, separated by a semicolon. The name of the application corresponds to the name of the directory, or can be declared using : `options("shinymanager.application" = "my-app")`
- **additional columns** : add others columns to retrieve the values server-side after authentication

See Also

[read_db_decrypt](#)

Examples

```
## Not run:

# Credentials data
credentials <- data.frame(
  user = c("shiny", "shinymanager"),
  password = c("azerty", "12345"),
  stringsAsFactors = FALSE
)

# you can use keyring package to set database key
library(keyring)
key_set("R-shinymanager-key", "obiwankenobi")

# Create the database
create_db(
  credentials_data = credentials,
  sqlite_path = "path/to/database.sqlite", # will be created
  passphrase = key_get("R-shinymanager-key", "obiwankenobi")
)

## End(Not run)
```

db-crypted

Read / Write crypted table from / to a SQLite database

Description

Read / Write crypted table from / to a SQLite database

Usage

```
write_db_encrypt(conn, value, name = "credentials", passphrase = NULL)
```

```
read_db_decrypt(conn, name = "credentials", passphrase = NULL)
```

Arguments

conn	A DBIConnection object, as returned by dbConnect .
value	A data.frame.
name	A character string specifying the unquoted DBMS table name.
passphrase	A secret passphrase to crypt the table inside the database

Value

a data.frame for read_db_decrypt.

See Also

[create_db](#)

Examples

```
# connect to database
conn <- DBI::dbConnect(RSQLite::SQLite(), dbname = ":memory:")

# write to database
write_db_encrypt(conn, value = head(iris), name = "iris", passphrase = "supersecret")

# read
read_db_decrypt(conn = conn, name = "iris", passphrase = "supersecret")

# with wrong passphrase
## Not run:
read_db_decrypt(conn = conn, name = "iris", passphrase = "forgotten")

## End(Not run)

# with DBI method you'll get a crypted blob
DBI::dbReadTable(conn = conn, name = "iris")

# add some users to database
## Not run:
conn <- DBI::dbConnect(RSQLite::SQLite(), dbname = "path/to/database.sqlite")

# update "credentials" table
current_user <- read_db_decrypt(
  conn,
  name = "credentials",
  passphrase = key_get("R-shinymanager-key", "obiwankenobi")
)
```

```

add_user <- data.frame(user = "new", password = "pwdToChange",
                      start = NA, expire = NA, admin = TRUE)

new_users <- rbind.data.frame(current_user, add_user)

write_db_encrypt(
  conn,
  value = new_users,
  name = "credentials",
  key_get("R-shinymanager-key", "obiwankenobi")
)

# update "pwd_mngt" table
pwd_mngt <- read_db_decrypt(
  conn,
  name = "pwd_mngt",
  passphrase = key_get("R-shinymanager-key", "obiwankenobi")
)

pwd_mngt <- rbind.data.frame(
  pwd_mngt,
  data.frame(user = "new", must_change = T, have_changed = F, date_change = "")
)

write_db_encrypt(
  conn,
  value = pwd_mngt,
  name = "pwd_mngt",
  passphrase = key_get("R-shinymanager-key", "obiwankenobi")
)

## End(Not run)

```

fab_button

Create a FAB button

Description

Create a fixed button in bottom right corner with additional button(s) in it

Usage

```
fab_button(..., inputId = NULL, icon = NULL, status = "default")
```

Arguments

...	HTML tags 'a' or 'button' or <code>actionButton</code> (with NULL labels).
inputId	Id for the FAB button (act like an <code>actionButton</code>).

icon	An icon for the main button.
status	Bootstrap status to apply to the main button.

Examples

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

  ui <- fluidPage(

    tags$h1("FAB button"),

    tags$p("FAB button:"),
    verbatimTextOutput(outputId = "res_fab"),

    tags$p("Logout button:"),
    verbatimTextOutput(outputId = "res_logout"),

    tags$p("Info button:"),
    verbatimTextOutput(outputId = "res_info"),

    fab_button(
      actionButton(
        inputId = "logout",
        label = NULL,
        tooltip = "Logout",
        icon = icon("sign-out")
      ),
      actionButton(
        inputId = "info",
        label = NULL,
        tooltip = "Information",
        icon = icon("info")
      ),
      inputId = "fab"
    )
  )

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

    output$res_fab <- renderPrint({
      input$fab
    })

    output$res_logout <- renderPrint({
      input$logout
    })

    output$res_info <- renderPrint({
      input$info
    })
  }
}
```

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

generate_pwd *Simple password generation*

Description

Simple password generation

Usage

```
generate_pwd(n = 1)
```

Arguments

n Number of password(s)

Value

a character

Examples

```
generate_pwd()  
generate_pwd(3)
```

module-authentication *Authentication module*

Description

Authentication module

Usage

```
auth_ui(id, status = "primary", tag_img = NULL, tag_div = NULL)  
auth_server(input, output, session, check_credentials, use_token = FALSE)
```


Arguments

<code>id</code>	Module's id.
<code>status</code>	Bootstrap status to use for the panel and the button. Valid status are: "default", "primary", "success", "warning", "danger".
<code>tag_img</code>	A <code>tags\$img</code> to be displayed on top of the authentication module.
<code>tag_div</code>	A <code>tags\$div</code> to be displayed on bottom of the authentication module.
<code>input</code> , <code>output</code> , <code>session</code>	Standard Shiny server arguments.
<code>check_credentials</code>	Function with two arguments (user, the username provided by the user and password, his/her password). Must return TRUE or FALSE. To use additionnals arguments, set them with <code>purrr::partial</code> (see examples).
<code>use_token</code>	Add a token in the URL to check authentication. Should not be used directly.

Value

A reactiveValues with 3 slots :

- **result** : logical, result of authentication.
- **user** : character, name of connected user.
- **user_info** : information about the user.

Examples

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

  # data.frame with credentials info
  credentials <- data.frame(
    user = c("fanny", "victor"),
    password = c("azerty", "12345"),
    comment = c("alsace", "auvergne"),
    stringsAsFactors = FALSE
  )

  # app
  ui <- fluidPage(

    # authentication module
    auth_ui(
      id = "auth",
      tag_img = tags$img(
        src = "https://www.r-project.org/logo/Rlogo.png", width = 100
      ),
      tag_div = tags$div(
        tags$p(
          "For any question, please contact ",
          tags$a(
```

```

        href = "mailto:someone@example.com?Subject=Shiny%20aManager",
        target="_top", "administrator"
      )
    )
  )
),

# result of authentication
verbatimTextOutput(outputId = "res_auth"),

# classic app
headerPanel('Iris k-means clustering'),
sidebarPanel(
  selectInput('xcol', 'X Variable', names(iris)),
  selectInput('ycol', 'Y Variable', names(iris),
              selected=names(iris)[[2]]),
  numericInput('clusters', 'Cluster count', 3,
               min = 1, max = 9)
),
mainPanel(
  plotOutput('plot1')
)
)

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

  # authentication module
  auth <- callModule(
    module = auth_server,
    id = "auth",
    check_credentials = check_credentials(credentials)
  )

  output$res_auth <- renderPrint({
    reactiveValuesToList(auth)
  })

  # classic app
  selectedData <- reactive({

    req(auth$result) # <---- dependency on authentication result

    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)
}

```

 module-password

New password module

Description

New password module

Usage

```
pwd_ui(id, tag_img = NULL, status = "primary")
```

```
pwd_server(input, output, session, user, update_pwd, validate_pwd = NULL,
           use_token = FALSE)
```

Arguments

<code>id</code>	Module's id.
<code>tag_img</code>	A <code>tags\$img</code> to be displayed on the authentication module.
<code>status</code>	Bootstrap status to use for the panel and the button. Valid status are: "default", "primary", "success", "warning", "danger".
<code>input, output, session</code>	Standard Shiny server arguments.
<code>user</code>	A <code>reactiveValues</code> with a slot <code>user</code> , referring to the user for whom the password is to be changed.
<code>update_pwd</code>	A function to perform an action when changing password is successful. Two arguments will be passed to the function: <code>user</code> (username) and <code>password</code> (the new password). Must return a list with at least a slot <code>result</code> with <code>TRUE</code> or <code>FALSE</code> , according if the update has been successful.
<code>validate_pwd</code>	A function to validate the password enter by the user. Default is to check for the password to have at least one number, one lowercase, one uppercase and be of length 6 at least.
<code>use_token</code>	Add a token in the URL to check authentication. Should not be used directly.

Examples

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

  ui <- fluidPage(
    tags$h2("Change password module"),
    actionButton(
      inputId = "ask", label = "Ask to change password"
    ),
    verbatimTextOutput(outputId = "res_pwd")
  )

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

    observeEvent(input$ask, {
      insertUI(
        selector = "body",
        ui = tags$div(
          id = "module-pwd",
          pwd_ui(id = "pwd")
        )
      )
    })

    output$res_pwd <- renderPrint({
      reactiveValuesToList(pwd_out)
    })

    pwd_out <- callModule(
      module = pwd_server,
      id = "pwd",
      user = reactiveValues(user = "me"),
      update_pwd = function(user, pwd) {
        # store the password somewhere
        list(result = TRUE)
      }
    )

    observeEvent(pwd_out$relog, {
      removeUI(selector = "#module-pwd")
    })
  }

  shinyApp(ui, server)
}
```

Description

Secure a Shiny application and manage authentication

Usage

```
secure_app(ui, ..., enable_admin = FALSE, head_auth = NULL,
           theme = NULL, language = "en")

secure_server(check_credentials, timeout = 15,
              session = shiny::getDefaultReactiveDomain())
```

Arguments

ui	UI of the application.
...	Arguments passed to auth_ui .
enable_admin	Enable or not access to admin mode, note that admin mode is only available when using SQLite backend for credentials.
head_auth	Tag or list of tags to use in the <head> of the authentication page (for custom CSS for example).
theme	Alternative Bootstrap stylesheet, default is to use readable, you can use themes provided by shinythemes. It will affect the authentication panel and the admin page.
language	Language to use for labels, supported values are : "en", "fr".
check_credentials	Function passed to auth_server .
timeout	Timeout session (minutes) before logout if sleeping. Default to 15. 0 to disable.
session	Shiny session.

Value

A reactiveValues containing informations about the user connected.

Note

A special input value will be accessible server-side with `input$shinymanager_where` to know in which step user is : authentication, application, admin or password.

Examples

```
if (interactive()) {

  # define some credentials
  credentials <- data.frame(
    user = c("shiny", "shinymanager"),
    password = c("azerty", "12345"),
    stringsAsFactors = FALSE
  )
}
```

```
library(shiny)
library(shinymanager)

ui <- fluidPage(
  tags$h2("My secure application"),
  verbatimTextOutput("auth_output")
)

# Wrap your UI with secure_app
ui <- secure_app(ui)

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

  # call the server part
  # check_credentials returns a function to authenticate users
  res_auth <- secure_server(
    check_credentials = check_credentials(credentials)
  )

  output$auth_output <- renderPrint({
    reactiveValuesToList(res_auth)
  })

  # your classic server logic

}

shinyApp(ui, server)
}
```

Index

auth_server, [13](#)
auth_server (module-authentication), [8](#)
auth_ui, [13](#)
auth_ui (module-authentication), [8](#)

check_credentials, [2](#)
create_db, [2, 3, 5](#)

db-encrypted, [4](#)
dbConnect, [5](#)

fab_button, [6](#)

generate_pwd, [8](#)

module-authentication, [8](#)
module-password, [11](#)

pwd_server (module-password), [11](#)
pwd_ui (module-password), [11](#)

read_db_decrypt, [4](#)
read_db_decrypt (db-encrypted), [4](#)

secure-app, [12](#)
secure_app (secure-app), [12](#)
secure_server (secure-app), [12](#)

write_db_encrypt (db-encrypted), [4](#)