Package 'shiny.reglog'

March 26, 2022

Title Optional Login and Registration Module System for ShinyApps

Version 0.5.0

Description RegLog system provides a set of shiny modules to handle register procedure for your users, alongside with login, edit credentials and password reset functionality.
It provides support for popular SQL databases
and optionally googlesheet-based database for easy setup. For email sending it provides support for 'emayili' and 'gmailr' backends. Architecture makes customizing usability pretty straightforward.
The authentication system created
with shiny.reglog is designed to be optional: user don't need to be logged-in to access your application, but when logged-in the user data can be used to read from and write to relational databases.

Depends R (>= 4.1.0), R6, shiny

Imports dplyr, lubridate, lifecycle, scrypt, shinyjs, stringi, uuid

Suggests covr, DBI, DT, devtools, emayili, gmailr, googledrive, googlesheets4, knitr, rmarkdown, RSQLite, testthat (>= 3.0.0)

Config/Needs/shinytesting devtools, shinytest, withr

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.1.2

VignetteBuilder knitr

Config/testthat/edition 3

URL https://statismike.github.io/shiny.reglog/

NeedsCompilation no

Author Michal Kosinski [aut, cre] (<https://orcid.org/0000-0002-8426-3654>)

Maintainer Michal Kosinski <kosinski.mich@gmail.com>

Repository CRAN

Date/Publication 2022-03-25 23:20:02 UTC

R topics documented:

`DBI_tables_create` *Create RegLog-valid database tables with DBI*

Description

Create RegLog-valid database tables with DBI

Usage

```
DBI_tables_create(
  conn,
  account_name = "account",
  reset_code_name = "reset_code",
  use_log = FALSE,
  log_name = "logs",
  user_data = NULL,
  hash_passwords = FALSE,
  verbose = TRUE
)
```

Arguments

`conn`	DBI connection object
`account_name`	Name of the table for storing user accounts credentials. Defaults to 'account'. Mandatory table.

`reset_code_name`
: Name of the table for storing generated password reset codes. Defaults to 'reset_code'. Mandatory table.

`use_log`
: Should the table for keeping RegLogServer logs be also created? Defaults to FALSE

`log_name`
: Name of the table for storing logs from RegLogServer object. Used only if `use_log = TRUE`. Defaults to `logs`

`user_data`
: If you wish to import existing user database, you can input data.frame with that table in this argument. It should contain columns: username, password, email. Defaults to NULL.

`hash_passwords`
: If you are importing table of users upon tables creation, you can also specify if the password should be hashed using `scrypt::hashPassword`. Defaults to `FALSE`. If you have unhashed passwords in imported table, set this option to `TRUE`.

`verbose`
: Boolean specific if the actions made by function should be printed back to the console. Defaults to `TRUE`.

Details

Currently, the function is tested and working correctly for SQLite, MySQL, MariaDB and PostgreSQL databases. If you want to use another DBI-supported database, you need to create tables in other ways.

Created tables should have following structure:

- account (default name)
  - id: integer, primary key, auto-increment
  - username: varchar(255), NOT NULL, unique
  - password: varchar(255), NOT NULL
  - email: varchar(255), NOT NULL, unique
  - create_time: datetime, NOT NULL
  - update_time: datetime, NOT NULL
- reset_code (default name)
  - id: integer, primary key, auto-increment
  - user_id: integer, NOT NULL
  - reset_code: varchar(10), NOT NULL
  - used: tinyint, NOT NULL
  - create_time: datetime, NOT NULL
  - update_time: datetime, NOT NULL
- logs (default name, optional)
  - id: integer, primary key, auto-increment
  - time: datetime, NOT NULL
  - session: varchar(255), NOT NULL
  - direction: varchar(255), NOT NULL
  - type: varchar(255), NOT NULL
  - note: varchar(255)

Value

List with results of the creation

See Also

Other RegLog databases: `gsheet_tables_create()`

Examples

```
library(shiny.reglog)

# create a tenporary SQLite database
conn <- DBI::dbConnect(
  RSQLite::SQLite(),
  dbname = ":memory:"
)

# mockup user data
user_data <-
  data.frame(username = c("Whatever", "Hanuka", "Helsinki", "How_come"),
             password = c("&f5*MSYj^niDt=V'3.[dyEX.C/", "%}&B[fs\\}5PKE@,*+V\\tx9\"at]",
                    "35z*ofW\\'G_8,@vCC`]~?e$Jm%", "s:;r_eLn?-D6;oA-=\"^R(-Ew<x"),
         email = c("what@mil.com", "hehe@soso.so", "nider@what.no", "crazzz@simpsy.com"))

# create the tables and input the data (hashing the passwords in the process)
DBI_tables_create(conn = conn,
                  user_data = user_data,
                  hash_passwords = TRUE,
                  verbose = FALSE)

# check generater tables
DBI::dbListTables(conn = conn)

# check the "user" table for user data
DBI::dbReadTable(conn = conn,
                 "account")

# disconnect
DBI::dbDisconnect(conn = conn)
```

---

db_timestamp *function to create standardized timestamp*

Description

function to create standardized timestamp

Usage

```
db_timestamp()
```

gsheet_tables_create *Create RegLog-valid database tables with googlesheets4*

Description

Create RegLog-valid database tables with googlesheets4

Usage

```
gsheet_tables_create(
  account_name = "account",
  reset_code_name = "reset_code",
  use_log = FALSE,
  log_name = "logs",
  user_data = NULL,
  hash_passwords = FALSE,
  gsheet_ss = NULL,
  gsheet_name = NULL,
  verbose = TRUE
)
```

Arguments

account_name	Name of the sheet for storing user accounts credentials. Defaults to 'account'. Mandatory spreadsheet.
reset_code_name	Name of the sheet for storing generated password reset codes. Defaults to 'reset_code'. Mandatory table.
use_log	Should the sheet for keeping RegLogServer logs be also created? Defaults to FALSE
log_name	Name of the sheet for storing logs from RegLogServer object. Used only if use_log = TRUE. Defaults to logs
user_data	If you wish to import existing user database, you can input data.frame with that table in this argument. It should contain columns: username, password, email. Defaults to NULL.
hash_passwords	If you are importing table of users upon tables creation, you can also specify if the password should be hashed using scrypt::hashPassword. Defaults to FALSE. If you have unhashed passwords in imported table, set this option to TRUE.
gsheet_ss	ID of the googlesheet that you want to append created tables to. Defaults to NULL, which means creating new googlesheet.
gsheet_name	If gsheet_ss = NULL and new googlesheet will be generated, you can choose choose its name. If left at default NULL, name will be generated randomly.
verbose	Boolean specific if the actions made by function should be printed back to the console. Defaults to TRUE. Don't affect googlesheets4 generated messages. To silence them, use options(googlesheets4_quiet = TRUE) in the script before.

Details

Created spreadsheets will have following structure:

- account (default name)
  - username: character
  - password: character
  - email: character
  - create_time: character
  - update_time: character
- reset_code (default name)
  - user_id: numeric
  - reset_code: character
  - used: numeric
  - create_time: character
  - update_time: character
- logs (default name, optional)
  - time: character
  - session: character
  - direction: character
  - type: character
  - note: character

Value

ID of the googlesheet

See Also

Other RegLog databases: `DBI_tables_create()`

Examples

```
if (googlesheets4::gs4_has_token()) {
  library(shiny.reglog)

  # mockup user data
  user_data <-
    data.frame(username = c("Whatever", "Hanuka", "Helsinki", "How_come"),
               password = c("&f5*MSYj^niDt=V'3.[dyEX.C/", "%}&B[fs\\}5PKE@,*+V\\tx9\"at]",
                            "35z*ofW\\'G_8,@vCC`]~?e$Jm%", "s:;r_eLn?-D6;oA-=\"^R(-Ew<x"),
          email = c("what@mil.com", "hehe@soso.so", "nider@what.no", "crazzz@simpsy.com"))

  # create the tables and input the data (hashing the passwords in the process)
  id <- gsheet_tables_create(user_data = user_data,
                             hash_passwords = TRUE,
                             verbose = FALSE)
```

```
  # check generated googlesheet
  googlesheets4::gs4_get(id)

  # check the "account" sheet for credentials data
  googlesheets4::read_sheet(id, "account")

  # remove example googlesheets
  googledrive::drive_trash(id)
}
```

`mailMessageAttachment` *Mail attachment data to be handled by mailConnector via* `custom_mail` *RegLogConnectorMessage*

Description

Mail attachment data to be handled by mailConnector via `custom_mail` RegLogConnectorMessage

Usage

```
mailMessageAttachment(filepath, filename = NULL, cid = NULL, filetype = NULL)
```

Arguments

`filepath`	path to the file to be attached
`filename`	name of the file to be used (supported by RegLogEmayiliConnector)
`cid`	content ID to be used to access in email body
`filetype`	mime type of the attached file

Value

mailMessageAttachment object

`RegLogConnector` *RegLog connector template*

Description

Parent class for all RegLog connectors

Public fields

`module_id` character vector specifying the automatically-generated module_id for listener server module

`listener` reactiveVal that the object keeps listening of changes for

`message` reactiveVal containing outward message

`log` list containing data about received and sent messages by the object

`handlers` named list containing functions used to handle different types of `RegLogConnectorMessage`. Name of the element corresponds to the 'type' that is should handle.

Methods

Public methods:

- `RegLogConnector$get_logs()`
- `RegLogConnector$new()`
- `RegLogConnector$suspend()`
- `RegLogConnector$resume()`
- `RegLogConnector$clone()`

Method `get_logs()`**:** Function to receive all saved logs from the object in the form of single data.frame

Usage:

```
RegLogConnector$get_logs()
```

Details: You can specify custom handler functions as a named list passed to `custom_handlers` arguments during object initialization. Custom handler should take arguments: `self` and `private` - relating to the R6 object and `message` of class `RegLogConnectorMessage`. It should return return `RegLogConnectorMessage` object

Returns: data.frame

Method `new()`**:** Initialization of the object. Sets up listener reactiveVal and initializes listening server module

Usage:

```
RegLogConnector$new(custom_handlers = NULL)
```

Arguments:

`custom_handlers` named list of custom handler functions. Custom handler should take arguments: `self` and `private` - relating to the R6 object and `message` of class `RegLogConnectorMessage`. It should return return `RegLogConnectorMessage` object.

Returns: object of `RegLogConnector` class

Method `suspend()`**:** Suspend the listening to the changes

Usage:

```
RegLogConnector$suspend()
```

Method `resume()`**:** Resume the listening to the changes

Usage:

```
RegLogConnector$resume()
```

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

Usage:

```
RegLogConnector$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`RegLogConnectorMessage` *create RegLogConnectorMessage object*

Description

Create an object of ReglogConnectorMessage class. It is used to send data to objects that inherit their class from `RegLogConnector`

Usage

```
RegLogConnectorMessage(type, ..., logcontent = NULL)
```

Arguments

`type`	character string declaring the type of message
`...`	named arguments that will be passed as data
`logcontent`	character string. Optional description to save into logs.

Value

object of `RegLogConnector` class, containing fields:

- *time*: numeric representation of `Sys.time()`
- *type*: character specifying the type of message
- *data*: list of values that are to be sent alongside the message
- *logcontent*: Character string with information to be saved in logs. Optional.

RegLogDBIConnector *Connector to DBI-valid databases*

Description

Object of this class handles all connections for the RegLogServer object to the database. It is created to handle DBI compatible drivers. Provides methods than will be used by RegLogServer to get and send data.

Super class

shiny.reglog::RegLogConnector -> RegLogDBIConnector

Methods

Public methods:

- RegLogDBIConnector$new()
- RegLogDBIConnector$clone()

Method new(): Initialization of the object. Creates initial connection to the database.

Usage:

```
RegLogDBIConnector$new(
  driver,
  ...,
  table_names = c("account", "reset_code", "logs"),
  custom_handlers = NULL
)
```

Arguments:

driver Call that specifies the driver to be used during all queries

... other arguments used in DBI::dbConnect() call

table_names character vector. Contains names of the tables in the database: first containing user data, second - reset codes information, third (optional) - logs from the object. For more info check documentation of DBI_database_create.

custom_handlers named list of custom handler functions. Custom handler should take arguments: self and private - relating to the R6 object and message of class RegLogConnectorMessage. It should return return RegLogConnectorMessage object.

Returns: object of RegLogDBIConnector class

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

Usage:

```
RegLogDBIConnector$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other dbConnectors: RegLogGsheetConnector

RegLogDemo *Demonstration ShinyApp with basic RegLog system*

Description

You can play a little with RegLogSever functionalities launching this ShinyApp. This demo needs also an installation of 'RSQLite' package to create and menage a temporary database.

Usage

```
RegLogDemo(emayili_smtp = NULL, emayili_from = NULL)
```

Arguments

emayili_smtp	defined emayili smtp server for you e-mail provider. If kept as default NULL, the e-mail sending functionality won't be used. If provided, it will require an installation of 'emayili' package.
emayili_from	String containing e-mail from which thesending will take place. Used only with 'emayili_smtp' defined.

RegLogEmayiliConnector

RegLogConnector for email sending via emayili *package*

Description

With the use of this object, RegLogServer can send emails confirming the registration and containing code for password reset procedure.

Super class

shiny.reglog::RegLogConnector -> RegLogEmayiliConnector

Public fields

mails List containing default mail templates to use by default mail handlers for register and password reset

Methods

Public methods:

- RegLogEmayiliConnector$new()
- RegLogEmayiliConnector$clone()

Method `new()`**:** Initialization of the object. Creates smtp server for email sending.

Usage:

```
RegLogEmayiliConnector$new(
  from,
  smtp,
  lang = "en",
  custom_txts = NULL,
  custom_handlers = NULL,
  custom_mails = NULL
)
```

Arguments:

`from` Character containing content in `from` of the email.

`smtp` Object created by `emayili::server` or all its similiar functions.

`lang` character specyfiyng which language to use for all texts generated in the UI. Defaults to 'en' for English. Currently 'pl' for Polish is also supported.

`custom_txts` named list containing character strings with custom messages. Defaults to NULL, so all built-in strings will be used.

`custom_handlers` named list of custom handler functions. Custom handler should take arguments: `self` and `private` - relating to the R6 object and `message` of class `RegLogConnectorMessage`. It should return return `RegLogConnectorMessage` object.

`custom_mails` named list containing character strings of the same structure as elements in the `mails` field. Not all elements need to be present.

Details: default mails are used by `register_mail` and `reset_pass_mail` handlers. To change the mail used by these handlers you can pass character strings to the `custom_mail` argument during initialization or append them directly into this list.
They are stored (and should be passed accordingly) in a list of structure:

- register
  - subject
  - body
- resetPass
  - subject
  - body
- credsEdit
  - subject
  - body

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

Usage:

```
RegLogEmayiliConnector$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other mailConnectors: RegLogGmailrConnector

RegLogGmailrConnector *RegLogConnector for email sending via* emayili *package*

Description

With the use of this object, RegLogServer can send emails confirming the registration and containing code for password reset procedure.

Super class

shiny.reglog::RegLogConnector -> RegLogGmailrConnector

Public fields

`mails` List containing default mail templates to use by default mail handlers for register and password reset

Methods

Public methods:

- RegLogGmailrConnector$new()
- RegLogGmailrConnector$clone()

Method `new()`**:** Initialization of the object. Creates smtp server for email sending.

Usage:

```
RegLogGmailrConnector$new(
  from,
  lang = "en",
  custom_txts = NULL,
  custom_handlers = NULL,
  custom_mails = NULL
)
```

Arguments:

`from` Character containing content in `from` of the email.

`lang` character specyfiyng which language to use for all texts generated in the UI. Defaults to 'en' for English. Currently 'pl' for Polish is also supported.

`custom_txts` named list containing character strings with custom messages. Defaults to NULL, so all built-in strings will be used.

`custom_handlers` named list of custom handler functions. Custom handler should take arguments: `self` and `private` - relating to the R6 object and `message` of class `RegLogConnectorMessage`. It should return return `RegLogConnectorMessage` object.

`custom_mails` named list containing character strings of the same structure as elements in the `mails` field. Not all elements need to be present.

Details: default mails are used by `register_mail` and `reset_pass_mail` handlers. To change the mail used by these handlers you can pass character strings to the `custom_mail` argument during initialization or append them directly into this list.
They are stored (and should be passed accordingly) in a list of structure:

- register
  - subject
  - body
- resetPass
  - subject
  - body
- credsEdit
  - subject
  - body

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

Usage:

```
RegLogGmailrConnector$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other mailConnectors: `RegLogEmayiliConnector`

`RegLogGsheetConnector` *Connector to googlesheet database*

Description

Object of this class handles all connections for the RegLogServer object to the database. It is created to handle googlesheet database. Provides methods than will be used by RegLogServer to get and send data.

Super class

`shiny.reglog::RegLogConnector` -> `RegLogGsheetConnector`

Methods

Public methods:

- RegLogGsheetConnector$new()
- RegLogGsheetConnector$clone()

Method `new()`**:** Initialization of the object. Creates initial connection to the database.

Usage:

```
RegLogGsheetConnector$new(
  gsheet_ss,
  gsheet_sheetnames = c("account", "reset_code", "logs"),
  custom_handlers = NULL
)
```

Arguments:

`gsheet_ss` id of the googlesheet holding database

`gsheet_sheetnames` character vector. Contains names of the sheets in the googlesheet: first containing user data, second - reset codes information, third (optional) - logs from the object. For more info check documentation of `gsheet_database_create`.

`custom_handlers` named list of custom handler functions. Custom handler should take arguments: `self` and `private` - relating to the R6 object and `message` of class `RegLogConnectorMessage`. It should return `RegLogConnectorMessage` object.

Returns: object of `RegLogDBIConnector` class

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

Usage:

```
RegLogGsheetConnector$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other dbConnectors: RegLogDBIConnector

`RegLogServer` *Login and registration moduleServer*

Description

RegLogServer is an R6 class to use for handling the whole backend of login and registration component of your shinyApp.

Public fields

`is_logged` reactiveVal containing logical indicating if the user is logged in

`user_id` reactiveVal containing character specifying the logged user name. If the user is not logged in, it will consist of uuid generated with `uuid::UUIDgenerate`

`user_mail` reactiveVal cantaining character string specifying the logged user mail. When not logged in, it contains NULL.

`account_id` reactiveVal caintaining integer specifying the logged user account's id number: for SQL database it is equal to the value contained withing `id` variable. For googlesheets database it is equal to the row number - 1 (the header). If not logged, it contains NULL.

`mail_message` reactiveVal containing most recent RegLogConnectorMessage received from mailConnector

`message` reactiveVal containing most recent RegLogConnectorMessage received from dbConnector or generated by RegLogServer itself.

`module_id` character storing ID for reglog_system module.

`dbConnector` `RegLogConnector` object used for communication with the database. Build-in children classes are `RegLogDBIConnector` and `RegLogGsheetConnector`.

`mailConnector` `RegLogConnector` object used for sending emails. Built-in children classes are `RegLogEmayiliConnector` and `RegLogGmailrConnector`.

`log` list containing all messages send and received.

`UI_list_login` reactiveVal holding the tagList of whole login UI.

`UI_list_resetPass` reactiveVal holding the tagList of whole resetPass UI.

`UI_list_credsEdit` reactiveVal holding the tagList of whole credentioals edit UI.

`UI_list_register` reactiveVal holding the tagList of whole register UI.

Methods

Public methods:

- `RegLogServer$new()`
- `RegLogServer$logout()`
- `RegLogServer$get_logs()`
- `RegLogServer$clone()`

Method `new()`**:** Initialize 'ReglogServer' moduleServer

Usage:

```
RegLogServer$new(
  dbConnector,
  mailConnector,
  app_name = basename(getwd()),
  app_address = NULL,
  lang = "en",
  custom_txts = NULL,
  use_modals = TRUE,
  module_id = "login_system"
)
```

Arguments:

`dbConnector` object of class `RegLogConnector` handling the reads from and writes to database. Two available in the package are `RegLogDBIConnector` and `RegLogGsheetsConnector`. See their documentation for more information about usage and creation of custom dbConnectors.

`mailConnector` object of class `RegLogConnector` handling the email sending to the user for register confirmation and password reset. Two available in the package are `RegLogEmayiliConnector` and `RegLogGmailrConnector`. See their documentation for more information about usage and creation of custom mailConnectors.

`app_name` Name of the app to refer during correspondence to users. Defaults to the name of working directory.

`app_address` URL to refer to during correspondence to users. If left at NULL, the URL will be parsed from `session$clientData`.

`lang` character specyfiyng which language to use for all texts generated in the UI. Defaults to 'en' for English. Currently 'pl' for Polish is also supported.

`custom_txts` named list containing character strings with custom messages. Defaults to NULL, so all built-in strings will be used.

`use_modals` either logical indicating if all (`TRUE`) or none (`FALSE`) modalDialogs should be shown or character vector indicating which modals should be shown. For more information see details.

`module_id` Character declaring the id of the module. Defaults to 'login_system'. Recommended to keep it that way, unless it would cause any namespace issues.

Method `logout()`**:** Method logging out logged user

Usage:

```
RegLogServer$logout()
```

Method `get_logs()`**:** Method to receive all saved logs from the object in the form of single data.frame

Usage:

```
RegLogServer$get_logs()
```

Returns: data.frame

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

Usage:

```
RegLogServer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Run only in interactive session #

if (interactive()) {

  library(shiny.reglog)
```

```
  # for exemplary setup temporary SQLite database will be created
  library("DBI")
  library("RSQLite")
  temp_sqlite <- tempfile(fileext = ".sqlite")
  conn <- DBI::dbConnect(RSQLite::SQLite(),
                         dbname = temp_sqlite)
  DBI_tables_create(conn)
  DBI::dbDisconnect(conn)

  # create minimalistic UI
  ui <- navbarPage(
    title = "RegLog system",
    tabPanel("Register", RegLog_register_UI("custom_id")),
    tabPanel("Login", RegLog_login_UI("custom_id")),
    tabPanel("Credentials edit", RegLog_credsEdit_UI("custom_id")),
    tabPanel("Password reset", RegLog_resetPass_UI("custom_id"))
  )

  # create server logic
  server <- function(input, output, session) {

    # create dbConnector with connection to the temporary SQLite database
    dbConnector <- RegLogDBIConnector$new(
      driver = RSQLite::SQLite(),
      dbname = temp_sqlite)

    # create mockup mailConnector
    mailConnector <- RegLogConnector$new()

    # create RegLogServer
    RegLog <- RegLogServer$new(
      dbConnector = dbConnector,
      mailConnector = mailConnector,
      ## all arguments below are optional! ##
      app_name = "RegLog example",
      app_address = "https://reglogexample.com",
      lang = "en",
      # custom texts as a named list with strings
      custom_txts = list(
        user_id = "Name of the user",
        register_success_t= "Congratulations - you have been registered in
                             successfully with RegLog system!"),
      # use modals as a named list of FALSE to inhibit specific modal
      use_modals = list(
        login_success = FALSE),
      # custom module id - provide the same to the UI elements!
      module_id = "custom_id")
  }

  shinyApp(ui, server)
}
```

RegLog_credsEdit_UI *Generate Edit User Data UI for RegLog system*

Description

Generate Edit User Data UI for RegLog system

Usage

```
RegLog_credsEdit_UI(module_id = "login_system")
```

Arguments

module_id	Character declaring the id of the module. Defaults to 'login_system'. Recommended to keep it that way, unless it would cause any namespace issues.

See Also

Other RegLog UI: RegLog_login_UI(), RegLog_register_UI(), RegLog_resetPass_UI()

RegLog_login_UI *Generate Login UI for RegLog system*

Description

Generate Login UI for RegLog system

Usage

```
RegLog_login_UI(module_id = "login_system")
```

Arguments

module_id	Character declaring the id of the module. Defaults to 'login_system'. Recommended to keep it that way, unless it would cause any namespace issues.

See Also

Other RegLog UI: RegLog_credsEdit_UI(), RegLog_register_UI(), RegLog_resetPass_UI()

RegLog_register_UI *Generate Register UI for RegLog system*

Description

Generate Register UI for RegLog system

Usage

```
RegLog_register_UI(module_id = "login_system")
```

Arguments

module_id	Character declaring the id of the module. Defaults to 'login_system'. Recommended to keep it that way, unless it would cause any namespace issues.

See Also

Other RegLog UI: RegLog_credsEdit_UI(), RegLog_login_UI(), RegLog_resetPass_UI()

RegLog_resetPass_UI *Generate ResetPass code UI for RegLog system*

Description

Generate ResetPass code UI for RegLog system

Usage

```
RegLog_resetPass_UI(module_id = "login_system")
```

Arguments

module_id	Character declaring the id of the module. Defaults to 'login_system'. Recommended to keep it that way, unless it would cause any namespace issues.

See Also

Other RegLog UI: RegLog_credsEdit_UI(), RegLog_login_UI(), RegLog_register_UI()

RegLog_txt *Getting texts for given language*

Description

Getting texts for given language

Usage

```
RegLog_txt(lang, x = NULL, custom_txts = NULL)
```

Arguments

lang	character to identify the language
x	character to identify the txt to get. If NULL, all labels are recovered
custom_txts	named list providing custom messages to replace default for specific languages.

Details

'RegLog_txt' outside of internal usage should be used only for getting the structure of all texts generated by 'shiny.reglog'.

To customize texts used by RegLog objects, provide within their call named list to the 'custom_txts' argument - it will be passed to 'custom_txts' within this call. You can check validity of your list by providing the 'custom_txts' and calling this function in console.

Values of list provided should be named in the same way as the default text you are willing to replace.

Index