Package: appReporteAvance (via r-universe)

September 1, 2024

Title Reporte de avances de oficina

Version 0.4.6.9000

Description Las personas de la oficina usan esta app para reportar el estado de avance de sus actividades específicas.

License MIT + file LICENSE

URL https://github.com/calderonsamuel/appReporteAvance

BugReports https://github.com/calderonsamuel/appReporteAvance/issues

Imports bs4Dash, bsicons, bslib, cli, colourpicker, config (>= 0.3.1), DBI, dplyr, firebase (>= 1.0.1.9000), fontawesome, fresh, glue, golem (>= 0.3.2), htmltools, ids, lubridate, methods, purrr, R6, RColorBrewer, reactable, readr, rlang, RMariaDB, sass, shiny (>= 1.7.1), shinyjs, shinyWidgets (>= 0.7.0), stringr, tibble, tidyr, writexl

Suggests spelling, testthat (>= 3.0.0)

Remotes JohnCoene/firebase

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

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

RemoteUrl https://github.com/calderonsamuel/appReporteAvance

RemoteRef HEAD

RemoteSha d469e0faa1ab2d9b11acc23b81bbd7c427a00d76

Contents

`alerts` *Pre styled sweet alerts*

Description

These functions are convenient wrappers for `shinyWidgets::sendSweetAlert()`.

Usage

```
alert_success(
  session = getDefaultReactiveDomain(),
  text = NULL,
  btn_labels = "Ok",
  btn_colors = "#3085d6",
  html = FALSE,
  closeOnClickOutside = TRUE,
  showCloseButton = FALSE,
  width = NULL,
  ...
)

alert_error(
  session = getDefaultReactiveDomain(),
  text = NULL,
  btn_labels = "Ok",
  btn_colors = "#3085d6",
```

```
  html = FALSE,
  closeOnClickOutside = TRUE,
  showCloseButton = FALSE,
  width = NULL,
  ...
)

alert_info(
  session = getDefaultReactiveDomain(),
  text = NULL,
  btn_labels = "Ok",
  btn_colors = "#3085d6",
  html = FALSE,
  closeOnClickOutside = TRUE,
  showCloseButton = FALSE,
  width = NULL,
  ...
)

alert_warning(
  session = getDefaultReactiveDomain(),
  text = NULL,
  btn_labels = "Ok",
  btn_colors = "#3085d6",
  html = FALSE,
  closeOnClickOutside = TRUE,
  showCloseButton = FALSE,
  width = NULL,
  ...
)
```

Arguments

session	The session object passed to function given to shinyServer.
text	Text of the alert.
btn_labels	Label(s) for button(s), can be of length 2, in which case the alert will have two buttons. Use NA for no buttons.s
btn_colors	Color(s) for the buttons.
html	Does text contains HTML tags ?
closeOnClickOutside	Decide whether the user should be able to dismiss the modal by clicking outside of it, or not.
showCloseButton	Show close button in top right corner of the modal.
width	Width of the modal (in pixel).
...	Other arguments passed to JavaScript method.

Value

HTML code for sweet alerts

See Also

shinyWidgets::sendSweetAlert()

AppData *Get App data*

Description

Get App data

Get App data

Details

R6 class that allows to get the information needed for an User session of the app.

Super classes

appReporteAvance::DBManager -> appReporteAvance::User -> appReporteAvance::Organisation -> appReporteAvance::Group -> appReporteAvance::Process -> appReporteAvance::Task -> AppData

Methods

Public methods:

- AppData$new()
- AppData$clone()

Method `new()`**:** Start a session of app data based on an user email

Usage:

```
AppData$new(email = Sys.getenv("REPORTES_EMAIL"))
```

Arguments:

`email` The email the user started the session with.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

buttons *Pre styled pretty action buttons*

Description

These functions have predefined styling and icons for labels. The defaults are inherited from `shinyWidgets::actionBttn()`.

Usage

```
btn_add(inputId, icon = NULL, size = "md", block = FALSE, no_outline = TRUE)

btn_trash(inputId, icon = NULL, size = "md", block = FALSE, no_outline = TRUE)

btn_agregar(
  inputId,
  icon = NULL,
  size = "md",
  block = FALSE,
  no_outline = TRUE
)

btn_guardar(
  inputId,
  icon = NULL,
  size = "md",
  block = FALSE,
  no_outline = TRUE
)

btn_modificar(
  inputId,
  icon = NULL,
  size = "md",
  block = FALSE,
  no_outline = TRUE
)

btn_cancelar(
  inputId,
  icon = NULL,
  size = "md",
  block = FALSE,
  no_outline = TRUE
)

btn_eliminar(
```

```
  inputId,
  icon = NULL,
  size = "md",
  block = FALSE,
  no_outline = TRUE
)

btn_minus(inputId, icon = NULL, size = "md", block = FALSE, no_outline = TRUE)

btn_refresh(
  inputId,
  icon = NULL,
  size = "md",
  block = FALSE,
  no_outline = TRUE
)

btn_expand(inputId, icon = NULL, size = "md", block = FALSE, no_outline = TRUE)

btn_user_add(
  inputId,
  icon = NULL,
  size = "md",
  block = FALSE,
  no_outline = TRUE
)

btn_user_remove(
  inputId,
  icon = NULL,
  size = "md",
  block = FALSE,
  no_outline = TRUE
)

btn_user_edit(
  inputId,
  icon = NULL,
  size = "md",
  block = FALSE,
  no_outline = TRUE
)

btn_editar(inputId, icon = NULL, size = "md", block = FALSE, no_outline = TRUE)
```

Arguments

inputId	The input slot that will be used to access the value.

icon	An optional icon to appear on the button.
size	Size of the button : xs,sm, md, lg.
block	Logical, full width button.
no_outline	Logical, don't show outline when navigating with keyboard/interact using mouse or touch.

Value

HTML code for pretty action buttons

See Also

shinyWidgets::actionBttn()

DBManager *Clase R6 para manejar la base de datos*

Description

Clase R6 para manejar la base de datos

Clase R6 para manejar la base de datos

Details

Una base de datos puede ejecutar y obtener querys

Public fields

con The DB connection

Methods

Public methods:

- DBManager$new()
- DBManager$db_execute_statement()
- DBManager$db_get_query()
- DBManager$db_make_query()
- DBManager$clone()

Method new(): Start the DB Manager

Usage:

```
DBManager$new(use_tibble = TRUE)
```

Arguments:

use_tibble Whether print the results of DbGetQuery() as a tibble or not. Default TRUE

Method `db_execute_statement()`**:** Analog to `DBI::dbExecute()`

Usage:

```
DBManager$db_execute_statement(...)
```

Arguments:

`...` Objects passed to glue::glue_sql() with the exception of the `.con` argument.

Method `db_get_query()`**:** Analog to `DBI::dbGetQuery()`

Usage:

```
DBManager$db_get_query(...)
```

Arguments:

`...` Objects passed to glue::glue_sql() with the exception of the `.con` argument.

Method `db_make_query()`**:** Construct the query to be passed to `DBI::dbGetQuery()`. Useful for debugging and subquery construction.

Usage:

```
DBManager$db_make_query(...)
```

Arguments:

`...` Objects passed to glue::glue_sql() with the exception of the `.con` argument.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`Group` *Get Group data*

Description

Get Group data

Get Group data

Details

R6 class that allows to get the Group information.

Super classes

`appReporteAvance::DBManager` -> `appReporteAvance::User` -> `appReporteAvance::Organisation` -> `Group`

Public fields

`group_selected` ID of the group to be used in the board, by default is the favorite group in group_users

Active bindings

`groups` List containing the group affiliations of the User

`group_users` List containing the user list of the group The info is shown following the User's group role.

`group_units` List containing the group's measurement units. Older units are shown first.

Methods

Public methods:

- `Group$new()`
- `Group$group_initialize()`
- `Group$group_add()`
- `Group$group_delete()`
- `Group$group_edit()`
- `Group$group_user_add()`
- `Group$group_user_delete()`
- `Group$group_user_edit()`
- `Group$group_select()`
- `Group$group_set_as_favorite()`
- `Group$group_unit_add()`
- `Group$group_unit_edit()`
- `Group$group_unit_delete()`
- `Group$clone()`

Method `new()`**:** Start a Group based on an user email

Usage:

```
Group$new(email = Sys.getenv("REPORTES_EMAIL"))
```

Arguments:

`email` The email the user started the session with.

Method `group_initialize()`**:** Initialize a group for a new user

Usage:

```
Group$group_initialize(org_id)
```

Arguments:

`org_id` The id of the organisation on which the statement will be executed

Method `group_add()`**:** Add a group to the database

Usage:

```
Group$group_add(org_id, group_title, group_description)
```

Arguments:

`org_id` The id of the organisation on which the statement will be executed

`group_title` The new title of the group

`group_description` The new description of the group

Method `group_delete()`**:** Remove a group from the database

Usage:

```
Group$group_delete(group_id)
```

Arguments:

`group_id` The id of the group on which the statement will be executed

Method `group_edit()`**:** Edit group metadata

Usage:

```
Group$group_edit(group_id, group_title, group_description)
```

Arguments:

`group_id` The id of the group on which the statement will be executed

`group_title` The new title of the group

`group_description` The new description of the group

Method `group_user_add()`**:** Add an user to a group

Usage:

```
Group$group_user_add(
  group_id,
  user_id,
  user_color = "white",
  group_role = "user"
)
```

Arguments:

`group_id` The id of the group on which the statement will be executed

`user_id` The id of the user on which the statement will be executed

`user_color` The color of the user's cards

`group_role` The role for the user in the group

Method `group_user_delete()`**:** Delete an user from a group

Usage:

```
Group$group_user_delete(group_id, user_id)
```

Arguments:

`group_id` The id of the group on which the statement will be executed

`user_id` The id of the user on which the statement will be executed

Method `group_user_edit()`**:** Edit the role of a user inside a group and related information

Usage:

```
Group$group_user_edit(group_id, user_id, user_color, group_role)
```

Arguments:

`group_id` The id of the group on which the statement will be executed

`user_id` The id of the user on which the statement will be executed

`user_color` The color of the user's cards

`group_role` The role for the user in the group

Method `group_select()`**:** Select a group for use in the board

Usage:

```
Group$group_select(group_id)
```

Arguments:

`group_id` The id of the group on which the statement will be executed

Method `group_set_as_favorite()`**:** Set selected group as favorite in the database

Usage:

```
Group$group_set_as_favorite()
```

Method `group_unit_add()`**:** Add a measurement unit for a group

Usage:

```
Group$group_unit_add(
  unit_title,
  unit_description = "",
  unit_type,
  unit_icon = "file"
)
```

Arguments:

`unit_title` The title of a group's measurement unit

`unit_description` The description of a group's measurement unit

`unit_type` The type of a group's measurement unit. One of ("report", "task")

`unit_icon` The icon of a group's measurement unit. Should be the compatible with fontawesome::fa().

Method `group_unit_edit()`**:** Edit a measurement unit from a group

Usage:

```
Group$group_unit_edit(
  unit_id,
  unit_title,
  unit_description,
  unit_type,
  unit_icon
)
```

Arguments:

`unit_id` The id of a group's measurement unit

`unit_title` The title of a group's measurement unit

`unit_description` The description of a group's measurement unit

`unit_type` The type of a group's measurement unit. One of ("report", "task")

`unit_icon` The icon of a group's measurement unit. Should be the compatible with fontawesome::fa().

Method `group_unit_delete()`**:** Delete a measurement unit from a group

Usage:

```
Group$group_unit_delete(unit_id)
```

Arguments:

`unit_id` The id of a group's measurement unit

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`iconPicker_dep` *Icon Picker Dependency*

Description

Create a dependency for the icon picker element.

Usage

```
iconPicker_dep()
```

Value

a html dependency named "iconPicker".

input_icon_picker *Icon Picker*

Description

Create an icon picker element.

Usage

```
input_icon_picker(
  inputId,
  label,
  selected = NULL,
  btn_class = "btn-outline-secondary"
)
```

Arguments

inputId	chr: the ID of the input element.
label	chr: the label of the icon picker.
selected	chr: the name of the selected fontawesome icon. Must have the form 'fas fa-name'.
btn_class	chr: the class of the button element.

Value

a dropdown list with icons and a hidden input to hold their values.

mk_alert *Sweet alerts factory*

Description

Sweet alerts factory

Usage

```
mk_alert(title = "Title", type = NULL)
```

Arguments

title	Title of the alert.
type	Type of the alert : info, success, warning or error.

Value

A function to generate a sweet alert with a particular title and type.

mk_btn *Pretty action buttons factory*

Description

Pretty action buttons factory

Usage

```
mk_btn(color, label, style = "jelly")
```

Arguments

color	Color of the button : default, primary, warning, danger, success, royal.
label	The contents of the button, usually a text label.
style	Style of the button, to choose between simple, bordered, minimal, stretch, jelly, gradient, fill, material-circle, material-flat, pill, float, unite.

Value

A function to generate action buttons with a particular styling

Organisation *Get Organisation data*

Description

Get Organisation data

Get Organisation data

Details

R6 class that allows to get the Organisation information.

Super classes

appReporteAvance::DBManager -> appReporteAvance::User -> Organisation

Active bindings

orgs List containing the organisation affiliations of the User

org_users List containing the user list of the organisation. The info is shown following the User's organisation role.

Methods

Public methods:

- `Organisation$new()`
- `Organisation$fetch_orgs()`
- `Organisation$org_initialize()`
- `Organisation$org_add()`
- `Organisation$org_delete()`
- `Organisation$org_edit()`
- `Organisation$org_user_add()`
- `Organisation$org_user_delete()`
- `Organisation$org_user_edit()`
- `Organisation$org_finalize()`
- `Organisation$clone()`

Method `new()`**:** Start an Organisation based on an user email

Usage:

```
Organisation$new(email)
```

Arguments:

`email` The email the user started the session with.

Method `fetch_orgs()`**:** Get the public data from the organisations the user is a member of

Usage:

```
Organisation$fetch_orgs()
```

Method `org_initialize()`**:** Initialize an organisation for a new user

Usage:

```
Organisation$org_initialize()
```

Method `org_add()`**:** Add a new organisation to the database

Usage:

```
Organisation$org_add(org_title, org_description)
```

Arguments:

`org_title` The new title of the organisation

`org_description` The new description of the organisation

Method `org_delete()`**:** Delete an organisation from the database

Usage:

```
Organisation$org_delete(org_id)
```

Arguments:

`org_id` The id of the organisation on which the statement will be executed

Method `org_edit()`**:** Edit Organisation metadata

Usage:

```
Organisation$org_edit(org_id, org_title, org_description)
```

Arguments:

`org_id` The id of the organisation on which the statement will be executed
`org_title` The new title of the organisation
`org_description` The new description of the organisation

Method `org_user_add()`**:** Add an user to an organisation

Usage:

```
Organisation$org_user_add(org_id, user_id, org_role)
```

Arguments:

`org_id` The id of the organisation on which the statement will be executed
`user_id` The id of the user on which the statement will be executed
`org_role` The role for the user in the organisation

Method `org_user_delete()`**:** Delete an user from an organisation

Usage:

```
Organisation$org_user_delete(org_id, user_id)
```

Arguments:

`org_id` The id of the organisation on which the statement will be executed
`user_id` The id of the user on which the statement will be executed

Method `org_user_edit()`**:** Edit the role of a user inside an organisation

Usage:

```
Organisation$org_user_edit(org_id, user_id, org_role)
```

Arguments:

`org_id` The id of the organisation on which the statement will be executed
`user_id` The id of the user on which the statement will be executed
`org_role` The role for the user in the organisation

Method `org_finalize()`**:** Remove the existence of an organisation

Usage:

```
Organisation$org_finalize(org_id)
```

Arguments:

`org_id` The id of the organisation on which the statement will be executed

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Process *Process Class Definition*

Description

This is a class definition for the Process class. It inherits from the Group class.

Super classes

appReporteAvance::DBManager -> appReporteAvance::User -> appReporteAvance::Organisation -> appReporteAvance::Group -> Process

Methods

Public methods:

- Process$new()
- Process$fetch_processes()
- Process$fetch_units()
- Process$process_add()
- Process$process_delete()
- Process$process_edit()
- Process$unit_add()
- Process$unit_edit()
- Process$unit_delete()
- Process$clone()

Method `new()`**:** This function initializes a Process object. It calls the initialize() method of the Group class to inherit its properties.

Usage:

```
Process$new(email)
```

Arguments:

`email` character string with user's email address

Method `fetch_processes()`**:** This function fetches all the processes that belong to a group using the group_selected property. It uses the glue and DBI libraries to execute an SQL query on the database connection stored in the self$con property. The results are transformed using purrr::pmap(list).

Usage:

```
Process$fetch_processes()
```

Returns: a list of processed retrieved from the database

Method `fetch_units()`**:** This function fetches all the units that belong to a process identified by its ID using the glue and DBI libraries to execute an SQL query on the database connection stored in the self$con property. The results are transformed using purrr::pmap(list).

Usage:

```
Process$fetch_units(process_id)
```

Arguments:

`process_id` integer with the ID of the process to retrieve units from

Returns: a list of units retrieved from the database

Method `process_add()`**:** This function adds a process to the database using SQL queries executed through the glue and DBI libraries. It uses the ids library to generate random IDs for the new process and stores it in the ID column of the processes table.

Usage:

```
Process$process_add(title, description = NA)
```

Arguments:

`title` character string with the title of the process to add

`description` character string with the description of the process (optional)

Returns: the ID of the newly added process

Method `process_delete()`**:** This function deletes a process from the database using an SQL query executed through the glue and DBI libraries.

Usage:

```
Process$process_delete(process_id)
```

Arguments:

`process_id` integer with the ID of the process to delete

Method `process_edit()`**:** This function edits the title and description of a process identified by its ID in the database. It uses SQL queries to update the title and description columns of the processes table executed through the glue and DBI libraries.

Usage:

```
Process$process_edit(process_id, title, description)
```

Arguments:

`process_id` integer with the ID of the process to edit

`title` character string with the new title for the process

`description` character string with the new description for the process

Method `unit_add()`**:** This function adds a measurement unit for a process identified by its ID to the database using SQL queries executed through the glue and DBI libraries. It uses the ids library to generate random IDs for the new unit and stores it in the ID column of the units table. It also checks if the unit type is either "report" or "task", and sets the "type" column of the units table accordingly.

Usage:

```
Process$unit_add(
  process_id,
  unit_title,
  unit_description = "",
  unit_type,
  unit_icon = "file"
)
```

Arguments:

`process_id` integer with the ID of the process to add a unit to

`unit_title` character string with the title of the unit to add

`unit_description` character string with the description of the unit (optional, defaults to "")

`unit_type` character string with the type of the unit, must be either "report" or "task"

`unit_icon` character string with the icon for the unit (optional, defaults to "file")

Returns: the ID of the newly added unit

Method `unit_edit()`**:** This function edits the title, description, icon and/or type of a measurement unit identified by its ID in the database. It uses SQL queries to update the title, description, icon and/or type columns of the units table executed through the glue and DBI libraries. It also checks if the unit type is either "report" or "task", and sets the "type" column of the units table accordingly.

Usage:

```
Process$unit_edit(unit_id, unit_title, unit_description, unit_type, unit_icon)
```

Arguments:

`unit_id` integer with the ID of the unit to be edited

`unit_title` character string with the new title for the unit

`unit_description` character string with the new description for the unit

`unit_type` character string with the new type of the unit, must be either "report" or "task"

`unit_icon` character string with the new icon for the unit

Method `unit_delete()`**:** This function deletes a measurement unit identified by its ID from the database using an SQL query executed through the glue and DBI libraries.

Usage:

```
Process$unit_delete(unit_id)
```

Arguments:

`unit_id` integer with the ID of the unit to delete

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

reportes_icon_names *Get Fontawesome Icons*

Description

Get the available solid style icons in the fontawesome package.

Usage

```
reportes_icon_names()
```

Value

a character vector with the names of the available icons.

run_app *Run the Shiny Application*

Description

Run the Shiny Application

Usage

```
run_app(
  onStart = NULL,
  options = list(),
  enableBookmarking = NULL,
  uiPattern = "/",
  ...
)
```

Arguments

onStart	A function that will be called before the app is actually run. This is only needed for `shinyAppObj`, since in the `shinyAppDir` case, a `global.R` file can be used for this purpose.
options	Named options that should be passed to the `runApp` call (these can be any of the following: "port", "launch.browser", "host", "quiet", "display.mode" and "test.mode"). You can also specify `width` and `height` parameters which provide a hint to the embedding environment about the ideal height/width for the app.
enableBookmarking	Can be one of `"url"`, `"server"`, or `"disable"`. The default value, `NULL`, will respect the setting from any previous calls to `enableBookmarking()`. See `enableBookmarking()` for more information on bookmarking your app.

uiPattern	A regular expression that will be applied to each GET request to determine whether the ui should be used to handle the request. Note that the entire request path must match the regular expression in order for the match to be considered successful.
...	arguments to pass to golem_opts. See ?golem::get_golem_options for more details.

Task *Get Group data*

Description

Get Group data

Get Group data

Details

R6 class that allows to get the Group information.

Super classes

appReporteAvance::DBManager -> appReporteAvance::User -> appReporteAvance::Organisation -> appReporteAvance::Group -> appReporteAvance::Process -> Task

Active bindings

tasks List containing the tasks an User can interact with

reports List containing the reports an User can interact with

Methods

Public methods:

- Task$new()
- Task$task_add()
- Task$task_delete()
- Task$task_edit()
- Task$task_report_progress()
- Task$task_archive()
- Task$progress_add()
- Task$task_get_history()
- Task$report_add()
- Task$report_delete()
- Task$report_archive()
- Task$fetch_reports_to_download()

- Task$clone()

Method `new()`**:** Start a Task based on an user email

Usage:

```
Task$new(email)
```

Arguments:

`email` The email the user started the session with.

Method `task_add()`**:** Add a new task for an User and report it as initial progress.

Usage:

```
Task$task_add(
  group_id,
  task_title,
  task_description,
  assignee,
  time_due,
  output_unit
)
```

Arguments:

`group_id` The id of the group on which the statement will be executed

`task_title` Title for the task

`task_description` Long description of the task

`assignee` The id of the user responsible for the task

`time_due` Deadline for the task completion. Datetime

`output_unit` Unit of measurement of the task output

Method `task_delete()`**:** Delete an user task

Usage:

```
Task$task_delete(task_id)
```

Arguments:

`task_id` The id of the task on which the statement will be executed

Method `task_edit()`**:** Edit a task metadata

Usage:

```
Task$task_edit(
  task_id,
  task_title = NULL,
  task_description = NULL,
  time_due = NULL
)
```

Arguments:

`task_id` The id of the task on which the statement will be executed

`task_title` Title for the task

`task_description` Long description of the task

`time_due` Deadline for the task completion. Datetime

Method `task_report_progress()`**:** Report progress on an assigned task

Usage:

```
Task$task_report_progress(task_id, status_current, details)
```

Arguments:

`task_id` The id of the task on which the statement will be executed
`status_current` Current status of the specified task
`details` Explanation of the progress made

Method `task_archive()`**:** Archive a task

Usage:

```
Task$task_archive(task_id)
```

Arguments:

`task_id` The id of the task on which the statement will be executed

Method `progress_add()`**:** Insert progress info on some task

Usage:

```
Task$progress_add(task_id, status, details)
```

Arguments:

`task_id` The id of the task on which the statement will be executed
`status` The status of the task once the new progress is added
`details` Explanation of the progress made

Method `task_get_history()`**:** Get a task's progression history

Usage:

```
Task$task_get_history(task_id)
```

Arguments:

`task_id` The id of the task on which the statement will be executed

Method `report_add()`**:** Add a report and the quantities it has contributed in the specified units

Usage:

```
Task$report_add(report_title, details, units, quantities)
```

Arguments:

`report_title` The title of the report
`details` Explanation of the progress made
`units` Units of the report. Must have the same size as quantities.
`quantities` Quantities of the report. Must have the same size as units.

Method `report_delete()`**:** Delete a report and its contributions from the database.

Usage:

```
Task$report_delete(report_id)
```

Arguments:

`report_id` The id of the report on which the statement will be executed

Method `report_archive()`**:** Archive a report and its contributions.

Usage:

```
Task$report_archive(report_id)
```

Arguments:

`report_id` The id of the report on which the statement will be executed

Method `fetch_reports_to_download()`**:** Get data for reporting

Usage:

```
Task$fetch_reports_to_download(start_date, end_date)
```

Arguments:

`start_date` Starting date of the report

`end_date` Ending date of the report

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`textAreaInputPro` *Create a text area input with enhanced parameters*

Description

Create a text area input with enhanced parameters

Usage

```
textAreaInputPro(
  inputId,
  label,
  value = "",
  width = NULL,
  height = NULL,
  cols = NULL,
  rows = NULL,
  placeholder = NULL,
  resize = NULL,
  maxlength = NULL,
  maxlengthCounter = FALSE,
  readonly = FALSE
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
value	Initial value.
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
height	The height of the input, e.g. '400px', or '100%'; see validateCssUnit().
cols	Value of the visible character columns of the input, e.g. 80. This argument will only take effect if there is not a CSS width rule defined for this element; such a rule could come from the width argument of this function or from a containing page layout such as fluidPage().
rows	The value of the visible character rows of the input, e.g. 6. If the height argument is specified, height will take precedence in the browser's rendering.
placeholder	A character string giving the user a hint as to what can be entered into the control. Internet Explorer 8 and 9 do not support this option.
resize	Which directions the textarea box can be resized. Can be one of "both", "none", "vertical", and "horizontal". The default, NULL, will use the client browser's default setting for resizing textareas.
maxlength	the maximum length of the input
maxlengthCounter	whether or not to show a character counter

Value

Returns a text area input with the specified parameters.

---

textInputPro *Create a text input with enhanced parameters*

Description

Create a text input with enhanced parameters

Usage

```
textInputPro(
  inputId,
  label,
  value = "",
  width = NULL,
  placeholder = NULL,
  maxlength = NULL,
  maxlengthCounter = FALSE,
  readonly = FALSE
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
value	Initial value.
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
placeholder	A character string giving the user a hint as to what can be entered into the control. Internet Explorer 8 and 9 do not support this option.
maxlength	the maximum length of the input
maxlengthCounter	whether or not to show a character counter

Value

Returns a text input with the specified parameters.

User *Get user information*

Description

Get user information

Get user information

Details

R6 class that allows to get the user information.

Super class

appReporteAvance::DBManager -> User

Public fields

user List containing the user info.

Methods

Public methods:

- User$new()
- User$user_add()
- User$user_delete()
- User$user_edit_names()
- User$clone()

Method `new()`**:** Start User

Usage:

```
User$new(email)
```

Arguments:

`email` The email the user started the session with.

Method `user_add()`**:** Add a new user to the database

Usage:

```
User$user_add(name, last_name, email)
```

Arguments:

`name` Name to be inserted as user metadata.
`last_name` Last name to be inserted as user metadata.
`email` The email the user started the session with.

Method `user_delete()`**:** Delete a user from the database

Usage:

```
User$user_delete(user_id)
```

Arguments:

`user_id` ID of the user to edit or delete.

Method `user_edit_names()`**:** Change the name and last name of the User

Usage:

```
User$user_edit_names(user_id, name, last_name)
```

Arguments:

`user_id` ID of the user to edit or delete.
`name` Name to be inserted as user metadata.
`last_name` Last name to be inserted as user metadata.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Index