Package ‘surveydown’

December 13, 2024

Title Markdown-Based Surveys Using 'Quarto' and 'shiny'

Version 0.7.2

Description Generate surveys using markdown and R code chunks. Surveys are composed of two files: a survey.qmd 'Quarto' file defining the survey content (pages, questions, etc), and an app.R file defining a 'shiny' app with global settings (libraries, database configuration, etc.) and server configuration options (e.g., conditional skipping / display, etc.). Survey data collected from respondents is stored in a 'PostgreSQL' database. Features include controls for conditional skip logic (skip to a page based on an answer to a question), conditional display logic (display a question based on an answer to a question), a customizable progress bar, and a wide variety of question types, including multiple choice (single choice and multiple choices), select, text, numeric, multiple choice buttons, text area, and dates. Because the surveys render into a 'shiny' app, designers can also leverage the reactive capabilities of 'shiny' to create dynamic and interactive surveys.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr

Depends R (>= 4.0.0)

Suggests knitr, leaflet, testthat, glue

Imports DBI, DT, fs, htmltools, markdown, pool, quarto, RPostgres, rstudioapi, rvest, shiny, shinyjs, shinyWidgets, usethis, utils, xml2, yaml

URL `https://pkg.surveydown.org`

BugReports `https://github.com/surveydown-dev/surveydown/issues`

NeedsCompilation no

Author John Paul Helveston [aut, cre, cph] (<`https://orcid.org/0000-0002-2657-9191`>), Pingfan Hu [aut, cph] (<`https://orcid.org/0009-0001-4877-4844`>), Bogdan Bunea [aut, cph] (<`https://orcid.org/0009-0006-2942-0588`>), Stefan Munnes [ctb]

Maintainer John Paul Helveston <`john.helveston@gmail.com`>

Repository CRAN

Date/Publication 2024-12-13 20:50:02 UTC

Contents

`sd_add_page` *Add a Page Template to the Current Document*

Description

This function inserts a template for a surveydown page at the current cursor position in the active RStudio document. It provides a basic structure for a new page, including a title, content area, and a next button. If the function call exists in the document, it will be removed before inserting the template.

Usage

```
sd_add_page()
```

Details

IMPORTANT: This function should be run outside any division or R code chunk in your 'Quarto' document. Running it inside a division or code chunk may result in an incorrect page structure.

The function performs the following steps:

1. Checks for and removes any existing `sd_add_page()` function call in the document.
2. Inserts a template at the current cursor position.

The template includes:

- A div with class `'sd-page'` and a placeholder page ID
- A placeholder for the page title
- A placeholder for page contents
- An R code chunk with a placeholder for questions and a next button

Value

This function does not return a value. It modifies the active document as a side effect by inserting text and potentially removing a function call.

Examples

```
if (interactive()) {
  library(surveydown)

  # Insert a new page template
  sd_add_page()
}
```

`sd_add_question` *Add a Question Template to the Current Document*

Description

This function inserts a template for a surveydown question at the current cursor position in the active RStudio document. It supports various question types and automatically removes the function call before inserting the template if it exists in the document.

Usage

```
sd_add_question(type = "mc", chunk = FALSE)
```

Arguments

`type`	A character string specifying the type of question template to insert. Default is `"mc"` (multiple choice). Available options are: • `"mc"`: Multiple choice (single selection) • `"select"`: Dropdown selection • `"mc_multiple"`: Multiple choice (multiple selections) • `"mc_buttons"`: Multiple choice with button layout (single selection) • `"mc_multiple_buttons"`: Multiple choice with button layout (multiple selections) • `"text"`: Short text input • `"textarea"`: Long text input • `"numeric"`: Numeric input • `"slider"`: Slider input • `"date"`: Date input • `"daterange"`: Date range input
`chunk`	Logical. If `TRUE`, the code will be generated with the R code chunk wrapper. Defaults to `FALSE`.

Details

The function performs the following steps:

1. Checks for and removes any existing `sd_add_question()` function call in the document.
2. Inserts the appropriate question template at the current cursor position.

Value

This function does not return a value. It modifies the active document as a side effect by inserting text and potentially removing a function call.

Examples

```
if (interactive()) {
  library(surveydown)

  # Insert a default multiple choice question template
  sd_add_question()

  # Insert a text input question template
  sd_add_question("text")

  # Insert a slider question template
  sd_add_question("slider")
}
```

`sd_close` *Create a 'Close' Button to Exit the Survey*

Description

This function creates a 'Close' button that, when clicked, will trigger the exit process for the survey. Depending on the server-side configuration, this may show a rating question or a simple confirmation dialog before attempting to close the current browser tab or window.

Usage

```
sd_close(label = NULL)
```

Arguments

`label`	Character string. The label of the 'Close' button. Defaults to `NULL`, in which case the word `"Exit Survey"` will be used.

Details

The function generates a 'shiny' action button that, when clicked, triggers the 'show_exit_modal' event. The server-side logic (controlled by the `rate_survey` parameter in `sd_server()`) determines whether to show a rating question or a simple confirmation dialog.

The function also includes a custom message handler for closing the window. This is necessary because some browsers may not allow JavaScript to close windows that were not opened by JavaScript. In such cases, the user will be prompted to close the tab manually.

Value

A 'shiny' tagList containing the 'Close' button UI element and associated JavaScript for the exit process.

Note

The actual behavior of the exit process (whether to show a rating question or not) is controlled by the `rate_survey` parameter in the `sd_server()` function, not in this UI function.

See Also

`sd_server`

Examples

```
if (interactive()) {
  library(surveydown)

  # Get path to example survey file
  survey_path <- system.file("examples", "sd_close.qmd",
                              package = "surveydown")

  # Copy to a temporary directory
  temp_dir <- tempdir()
  file.copy(survey_path, file.path(temp_dir, "survey.qmd"))
  orig_dir <- getwd()
  setwd(temp_dir)

  # Define a minimal server
  server <- function(input, output, session) {
    sd_server()
  }

  # Run the app
  shiny::shinyApp(ui = sd_ui(), server = server)

  # Clean up
  setwd(orig_dir)
}
```

sd_completion_code *Generate a Random Completion Code*

Description

This function generates a random completion code with a specified number of digits. The code is returned as a character string.

Usage

```
sd_completion_code(digits = 6)
```

Arguments

digits	An integer specifying the number of digits in the completion code. Must be a positive integer. Default is 6.

Value

A character string representing the random completion code.

Examples

```
library(surveydown)

sd_completion_code()  # generates a 6-digit code
sd_completion_code(digits = 8)  # generates an 8-digit code
sd_completion_code(digits = 4)  # generates a 4-digit code
sd_completion_code(digits = 10)  # generates a 10-digit code
```

`sd_copy_value` *Create a copy of a value*

Description

This function creates a copy of an input value and makes it available as a new output. The new output can then be displayed using `sd_output()`.

Usage

```
sd_copy_value(id, id_copy)
```

Arguments

`id`	Character string. The ID of the input value to copy.
`id_copy`	Character string. The ID for the new copy (must be different from `id`).

Value

`NULL` invisibly. This function is called for its side effects.

See Also

`sd_output()` for displaying the copied value

Examples

```
if (interactive()) {
  library(surveydown)

  # Get path to example survey file
  survey_path <- system.file("examples", "sd_ui.qmd",
                             package = "surveydown")

  # Copy to a temporary directory
  temp_dir <- tempdir()
  file.copy(survey_path, file.path(temp_dir, "sd_copy_value.qmd"))
  orig_dir <- getwd()
  setwd(temp_dir)

  # Define a minimal server
  server <- function(input, output, session) {

    # Make a copy of the "name" variable to call its value a second time
    sd_copy_value(id = "name", id_copy = "name_copy")

    sd_server()
  }
  # Run the app
  shiny::shinyApp(ui = sd_ui(), server = server)

  # Clean up
  setwd(orig_dir)
}
```

`sd_create_survey` *Create a new survey template*

Description

This function creates a new survey template by copying files from the package's template directory to a specified path. It handles file conflicts and provides appropriate warnings and feedback.

Usage

```
sd_create_survey(path = getwd(), structure = "single")
```

Arguments

`path`	A character string specifying the directory where the survey template should be created. Defaults to the current working directory.
`structure`	A character string specifying the template structure to use. Must be either `"single"` or `"multi"`. Defaults to `"single"`.

Details

The function performs the following steps:

- If the specified path is the current working directory, it asks for user confirmation.
- Validates the specified structure (`"single"` or `"multi"`).
- Creates the target directory if it doesn't exist.
- Copies all files from the package's template directory (based on the specified structure) to the target path.
- Preserves the directory structure of the template.
- Skips existing files and provides warnings for each skipped file.
- Handles .Rproj files specially, skipping if any .Rproj file already exists in the target directory.
- Provides feedback on whether files were copied or if all files already existed.

Value

Invisible `NULL`. The function is called for its side effects of creating files and providing user feedback.

Examples

```
if (interactive()) {
  # Create a single-page survey template
  sd_create_survey(structure = "single")

  # Create a multi-page survey template
  sd_create_survey(structure = "multi")
}
```

---

sd_create_translations

Create a translations template file

Description

This function creates a template translations.yml file in the project root directory that users can customize to modify system messages.

Usage

```
sd_create_translations(language = "en", path = getwd())
```

Arguments

`language`	Character string specifying the language to use. See https://shiny.posit.co/r/reference/shiny/1.7.0/dateinpu for supported languages. Also, if `"en"`, `"de"`, `"es"`, `"fr"`, or `"it"` is chosen, default messages in those langauges will be used, otherwise the default English messages will be used. Defaults to `"en"`.
`path`	Character string specifying the directory where the translations.yml file should be created. Defaults to the current working directory. The file should be placed in the root project folder of your surveydown survey.

Value

Invisible `NULL`.

Examples

```
if (interactive()) {
  # Create English template
  sd_create_translations()

  # Create German template
  sd_create_translations(language = "de")

  # Create Japanese template
  # Will use English messages but Japanese date picker - user can modify
  # the messages as desired
  sd_create_translations(language = "ja")
}
```

---

sd_database *Connect to a 'PostgreSQL' Database with Automatic Cleanup*

Description

This function establishes a connection pool to a 'PostgreSQL' database (e.g. Supabase) and sets up automatic cleanup when the 'shiny' session ends.

Usage

```
sd_database(
  host = NULL,
  dbname = NULL,
  port = NULL,
  user = NULL,
  table = NULL,
  password = Sys.getenv("SURVEYDOWN_PASSWORD"),
  gssencmode = "prefer",
  ignore = FALSE,
  min_size = 1,
  max_size = Inf
)
```

Arguments

`host`	Character string. The host address of the Supabase database.
`dbname`	Character string. The name of the Supabase database.
`port`	Integer. The port number for the Supabase database connection.
`user`	Character string. The username for the Supabase database connection.
`table`	Character string. The name of the table to interact with in the Supabase database.
`password`	Character string. The password for the Supabase database connection. NOTE: While you can provide a hard-coded password here, we do NOT recommend doing so for security purposes. Instead, you should establish a password with `surveydown::sd_set_password()`, which will create a local `.Renviron` file that stores your password as a `SURVEYDOWN_PASSWORD` environment variable. The `password` argument uses this as the default value, so if you set a password properly with `surveydown::sd_set_password()`, then you can safely ignore using the `password` argument here.
`gssencmode`	Character string. The GSS encryption mode for the database connection. Defaults to `"prefer"`. NOTE: If you have verified all connection details are correct but still cannot access the database, consider setting this to `"disable"`. This can be necessary if you're on a secure connection, such as a VPN.

`ignore`	Logical. If `TRUE`, data will be saved to a local CSV file instead of the database. Defaults to `FALSE`.
`min_size`	Integer. The minimum number of connections in the pool. Defaults to 1.
`max_size`	Integer. The maximum number of connections in the pool. Defaults to `Inf`.

Value

A list containing the database connection pool (`db`) and the table name (`table`), or `NULL` if in ignore mode or if there's an error.

Examples

```
if (interactive()) {
  # Assuming SURVEYDOWN_PASSWORD is set in .Renviron
  db <- sd_database(
    host   = "aws-0-us-west-1.pooler.supabase.com",
    dbname = "postgres",
    port   = "6---",
    user   = "postgres.k----------i",
    table  = "your-table-name",
    ignore = FALSE
  )

  # Print the structure of the connection
  str(db)

  # Close the connection pool when done
  if (!is.null(db)) {
    pool::poolClose(db$db)
  }
}
```

`sd_display_question` *Create a placeholder for a reactive survey question*

Description

This function is depreciated - use `sd_output()` instead.

Usage

```
sd_display_question(id)
```

Arguments

`id`	A unique identifier for the question.

Value

A 'shiny' UI element that serves as a placeholder for the reactive question.

`sd_display_value` *Display the value of a survey question*

Description

This function is depreciated - use `sd_output()` instead.

Usage

```
sd_display_value(id, display_type = "inline", wrapper = NULL, ...)
```

Arguments

`id`	The ID of the question to display
`display_type`	The type of display. Can be `"inline"` (default), `"text"`, `"verbatim"`, or `"ui"`.
`wrapper`	A function to wrap the output
`...`	Additional arguments passed to the wrapper function

Value

A 'shiny' UI element displaying the question's value

`sd_get_data` *Fetch data from a database table with automatic reactivity detection*

Description

This function retrieves all data from a specified table in a database. It automatically detects whether it's being used in a reactive context (e.g., within a 'shiny' application) and behaves accordingly. In a reactive context, it returns a reactive expression that automatically refreshes the data at specified intervals.

Usage

```
sd_get_data(db, refresh_interval = NULL)
```

Arguments

`db`	A list containing database connection details created using `sd_database()`. Must have elements: • `db`: A `DBI` database connection object • `table`: A string specifying the name of the table to query
`refresh_interval`	Numeric. The time interval (in seconds) between data refreshes when in a reactive context. Default is `NULL`, meaning the data will not refresh.

Value

In a non-reactive context, returns a data frame containing all rows and columns from the specified table. In a reactive context, returns a reactive expression that, when called, returns the most recent data from the specified database table.

Examples

```
# Non-reactive context example
## Not run:
  library(surveydown)

  # Assuming you have a database connection called db created using
  # sd_database(), you can fetch data with:

  data <- sd_get_data(db)
  head(data)

  # Reactive context example (inside a surveydown app)

  server <- function(input, output, session) {
    data <- sd_get_data(db, refresh_interval = 10)

    output$data_table <- renderTable({
      data()  # Note the parentheses to retrieve the reactive value
    })
  }

## End(Not run)
```

`sd_get_url_pars` *Get URL Parameters in a 'shiny' Application*

Description

This function retrieves URL parameters from the current 'shiny' session. It must be called from within a 'shiny' reactive context.

Usage

```
sd_get_url_pars(...)
```

Arguments

`...`	Optional. Names of specific URL parameters to retrieve. If none are specified, all URL parameters are returned.

Value

A reactive expression that returns a list of URL parameters.

Examples

```
if (interactive()) {
  library(surveydown)

  # Get path to example survey file
  survey_path <- system.file("examples", "sd_redirect.qmd",
                             package = "surveydown")

  # Copy to a temporary directory
  temp_dir <- tempdir()
  file.copy(survey_path, file.path(temp_dir, "survey.qmd"))
  orig_dir <- getwd()
  setwd(temp_dir)

  # Define a minimal server
  server <- function(input, output, session) {

    # Reactive expression that generates a url with an id variable
    # parsed from the url
    url_redirect <- reactive({
      params <- sd_get_url_pars()
      id <- params["id"]
      return(paste0("https://www.google.com?id=", id))
    })

    # Create the redirect button
    sd_redirect(
      id = "redirect_url_pars",
      url = url_redirect(),
      button = TRUE,
      label = "Redirect"
    )

    sd_skip_if(
      input$screening_question == "end_1" ~ "end_page_1",
      input$screening_question == "end_1" ~ "end_page_2",
    )

    sd_server()
  }

  # Run the app
  shiny::shinyApp(ui = sd_ui(), server = server)

  # Clean up
  setwd(orig_dir)
}
```

`sd_include_folder` *Include a folder to the 'shiny' resource path*

Description

This function includes a specified folder to the 'shiny' resource path, making it accessible for serving static files in a 'shiny' application. It checks for pre-existing resource paths to avoid conflicts with folders already included by the package.

Usage

```
sd_include_folder(folder)
```

Arguments

`folder`	A character string specifying the name of the folder to include. This folder should exist in the root directory of your 'shiny' app.

Value

`NULL` invisibly. The function is called for its side effect of adding a resource path to 'shiny'.

Examples

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

  # Create an "images" folder
  dir.create("images")

  # Include the folder in the shiny resource path
  sd_include_folder("images")
}
```

`sd_is_answered` *Check if a question is answered*

Description

This function checks if a given question has been answered by the user. For matrix questions, it checks if all sub-questions (rows) are answered.

Usage

```
sd_is_answered(question_id)
```

Arguments

`question_id`	The ID of the question to check.

Value

A logical value: TRUE if the question is answered, FALSE otherwise.

Examples

```
if (interactive()) {
  library(surveydown)

  # Get path to example survey file
  survey_path <- system.file("examples", "sd_is_answered.qmd",
                             package = "surveydown")

  # Copy to a temporary directory
  temp_dir <- tempdir()
  file.copy(survey_path, file.path(temp_dir, "survey.qmd"))
  orig_dir <- getwd()
  setwd(temp_dir)

  # Define a minimal server
  server <- function(input, output, session) {

    sd_show_if(
      # If "apple_text" is answered, show the conditional question
      sd_is_answered("apple_text") ~ "other_fruit"
    )

    sd_server()
  }

  # Run the app
  shiny::shinyApp(ui = sd_ui(), server = server)

  # Clean up
  setwd(orig_dir)
}
```

---

sd_next *Create a 'Next' Button for Page Navigation*

Description

This function creates a 'Next' button for navigating to the specified next page in a Surveydown survey. The button can be activated by clicking or by pressing the Enter key when visible.

Usage

```
sd_next(next_page = NULL, label = NULL)
```

Arguments

`next_page`	Character string. The ID of the next page to navigate to. This parameter is required.
`label`	Character string. The label of the 'Next' button. Defaults to `NULL`, in which case the word `"Next"` will be used.

Details

The function generates a 'shiny' action button that, when clicked or when the Enter key is pressed, sets the input value to the specified next page ID, facilitating page navigation within the Shiny application. The button is styled to appear centered on the page and includes a class for Enter key functionality.

Value

A 'shiny' tagList containing the 'Next' button UI element.

Examples

```
if (interactive()) {
  library(surveydown)

  # Get path to example survey file
  survey_path <- system.file("examples", "sd_next.qmd",
                             package = "surveydown")

  # Copy to a temporary directory
  temp_dir <- tempdir()
  file.copy(survey_path, file.path(temp_dir, "survey.qmd"))
  orig_dir <- getwd()
  setwd(temp_dir)

  # Define a minimal server
  server <- function(input, output, session) {
    sd_server()
  }

  # Run the app
  shiny::shinyApp(ui = sd_ui(), server = server)

  # Clean up
  setwd(orig_dir)
}
```

`sd_output` *Output Function for Displaying reactive objects and values*

Description

Output Function for Displaying reactive objects and values

Usage

```
sd_output(
  id,
  type = NULL,
  width = "100%",
  display = "text",
  inline = TRUE,
  wrapper = NULL,
  ...
)
```

Arguments

`id`	Character string. A unique identifier for the output element.
`type`	Character string. Specifies the type of output. Can be `"question"`, `"value"`, or `NULL`. If `NULL`, the function behaves like `shiny::uiOutput()`.
`width`	Character string. The width of the UI element. Defaults to `"100%"`.
`display`	Character string. Specifies the display type for `"value"` outputs. Can be `"text"`, `"verbatim"`, or `"ui"`. Only used when `type = "value"`.
`inline`	Logical. Whether to render the output inline. Defaults to `TRUE`.
`wrapper`	Function. A function to wrap the output. Only used when `type = "value"`.
`...`	Additional arguments passed to the underlying 'shiny' functions or the wrapper function.

Details

The function behaves differently based on the `type` parameter:

- If `type` is `NULL`, it acts like `shiny::uiOutput()`.
- If `type` is `"question"`, it creates a placeholder for a reactive survey question.
- If `type` is `"value"`, it creates an output to display the value of a survey question, with the display style determined by the `display` parameter.

Value

A 'shiny' UI element, the type of which depends on the input parameters.

Examples

```
if (interactive()) {
  library(surveydown)

  # Get path to example survey file
  survey_path <- system.file("examples", "sd_output.qmd",
                             package = "surveydown")

  # Copy to a temporary directory
  temp_dir <- tempdir()
  file.copy(survey_path, file.path(temp_dir, "survey.qmd"))
  orig_dir <- getwd()
  setwd(temp_dir)

  # Define a minimal server
  server <- function(input, output, session) {
    sd_server()
  }

  # Run the app
  shiny::shinyApp(ui = sd_ui(), server = server)

  # Clean up
  setwd(orig_dir)
}
```

`sd_question` *Create a survey question*

Description

This function creates various types of survey questions for use in a Surveydown survey.

Usage

```
sd_question(
  type,
  id,
  label,
  cols = "80",
  direction = "horizontal",
  status = "default",
  width = "100%",
  height = NULL,
  selected = NULL,
  label_select = "Choose an option...",
  grid = TRUE,
  individual = TRUE,
  justified = FALSE,
  force_edges = TRUE,
  option = NULL,
  placeholder = NULL,
  resize = NULL,
  row = NULL
)
```

Arguments

`type`	Specifies the type of question. Possible values are "select", "mc", "mc_multiple", "mc_buttons", "mc_multiple_buttons", "text", "textarea", "numeric", "slider", "date", "daterange", and "matrix".
`id`	A unique identifier for the question, which will be used as the variable name in the resulting survey data.
`label`	Character string. The label for the UI element, which can be formatted with markdown.
`cols`	Integer. Number of columns for the textarea input. Defaults to 80.
`direction`	Character string. The direction for button groups ("horizontal" or "vertical"). Defaults to "horizontal".
`status`	Character string. The status for button groups. Defaults to "default".
`width`	Character string. The width of the UI element. Defaults to "100%".
`height`	Character string. The height of the textarea input. Defaults to "100px".
`selected`	Value. The selected value(s) for certain input elements.
`label_select`	Character string. The label for the select input. Defaults to "Choose an option...".
`grid`	Logical. Whether to show a grid for slider input. Defaults to TRUE.
`individual`	Logical. Whether buttons in a group should be individually styled. Defaults to TRUE.
`justified`	Logical. Whether buttons in a group should fill the width of the parent div. Defaults to FALSE.
`force_edges`	Logical. Whether to force edges for slider input. Defaults to TRUE.
`option`	List. Options for the select, radio, checkbox, and slider inputs.
`placeholder`	Character string. Placeholder text for text and textarea inputs.
`resize`	Character string. Resize option for textarea input. Defaults to NULL.
`row`	List. Used for "matrix" type questions. Contains the row labels and their corresponding IDs.

Details

The function supports various question types:

- "select": A dropdown selection

- "mc": Multiple choice (single selection)
- "mc_multiple": Multiple choice (multiple selections allowed)
- "mc_buttons": Multiple choice with button-style options (single selection)
- "mc_multiple_buttons": Multiple choice with button-style options (multiple selections allowed)
- "text": Single-line text input
- "textarea": Multi-line text input
- "numeric": Numeric input
- "slider": Slider input
- "date": Date input
- "daterange": Date range input
- "matrix": Matrix-style question with rows and columns

For "matrix" type questions, use the `row` parameter to define the rows of the matrix. Each element in the `row` list should have a name (used as the row ID) and a value (used as the row label).

Value

A 'shiny' UI element wrapped in a div with a data attribute for question ID.

Examples

```
if (interactive()) {
  library(surveydown)

  # Get path to example survey file
  survey_path <- system.file("examples", "basic_survey.qmd",
                             package = "surveydown")

  # Copy to a temporary directory
  temp_dir <- tempdir()
  file.copy(survey_path, file.path(temp_dir, "survey.qmd"))
  orig_dir <- getwd()
  setwd(temp_dir)

  # Define a minimal server
  server <- function(input, output, session) {
    sd_server()
  }

  # Run the app
  shiny::shinyApp(ui = sd_ui(), server = server)

  # Clean up
  setwd(orig_dir)
}
```

`sd_question_custom` *Create a Custom Question with a Shiny Widget*

Description

This function creates a custom survey question that incorporates any Shiny widget and captures its interaction value. It allows for the integration of interactive visualizations (e.g., maps, plots) or other custom Shiny outputs into a survey, storing the result of user interaction as survey data.

Usage

```
sd_question_custom(id, label, output, value, height = "400px")
```

Arguments

`id`	Character string. A unique identifier for the question.
`label`	Character string. The label text for the question, which can include HTML formatting.
`output`	Shiny UI element. The output of a Shiny widget (e.g., `leafletOutput()`, `plotlyOutput()`).
`value`	Reactive expression that returns the value to be stored in the survey data when the user interacts with the widget.
`height`	Character string. The height of the widget output. Defaults to "400px".

Details

The function creates a custom question container that includes:

- A visible widget output that users can interact with
- A hidden text input that stores the value from the interaction
- Automatic tracking of user interaction for progress monitoring

The value to be stored is controlled by the reactive expression provided to the `value` parameter, which should update whenever the user interacts with the widget in the desired way.

Value

None (called for side effects)

See Also

`sd_question()` for standard question types

Examples

```
if (interactive()) {
  library(surveydown)
  library(leaflet)

  server <- function(input, output, session) {
    # Create map output
    output$usa_map <- renderLeaflet({
      leaflet() |>
        addTiles() |>
        setView(lng = -98.5795, lat = 39.8283, zoom = 4)
    })

    # Reactive value for selected location
    selected_location <- reactiveVal(NULL)

    # Click observer
    observeEvent(input$usa_map_click, {
      click <- input$usa_map_click
      if (!is.null(click)) {
        selected_location(
          sprintf("Lat: %0.2f, Lng: %0.2f", click$lat, click$lng)
        )
      }
    })

    # Create the custom question
    sd_question_custom(
      id = "location",
      label = "Click on your location:",
      output = leafletOutput("usa_map", height = "400px"),
      value = selected_location
    )

    sd_server()
  }

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

`sd_redirect` *Create a Redirect Element for 'shiny' Applications*

Description

This function creates a UI element that redirects the user to a specified URL. It can be used in both reactive and non-reactive contexts within 'shiny' applications.

Usage

```
sd_redirect(
  id,
  url,
  button = TRUE,
  label = "Click here",
  delay = NULL,
  newtab = FALSE
)
```

Arguments

`id`	A character string of a unique id to be used to identify the redirect button in the survey body.
`url`	A character string specifying the URL to redirect to.
`button`	A logical value indicating whether to create a button (`TRUE`) or a text element (`FALSE`) for the redirect. Default is `TRUE`.
`label`	A character string for the button or text label. Defaults to `NULL`, in which case the words `"Click here"` will be used.
`delay`	An optional numeric value specifying the delay in seconds before automatic redirection. If `NULL` (default), no automatic redirection occurs.
`newtab`	A logical value indicating whether to open the URL in a new tab (`TRUE`) or in the current tab (`FALSE`). Default is `FALSE`.

Value

In a reactive context, returns a function that when called, renders the redirect element. In a non-reactive context, returns the redirect element directly.

Examples

```
if (interactive()) {
  library(surveydown)

  # Get path to example survey file
  survey_path <- system.file("examples", "sd_redirect.qmd",
                             package = "surveydown")

  # Copy to a temporary directory
  temp_dir <- tempdir()
  file.copy(survey_path, file.path(temp_dir, "survey.qmd"))
  orig_dir <- getwd()
  setwd(temp_dir)

  # Define a minimal server
  server <- function(input, output, session) {

    # Reactive expression that generates a url with an id variable
    # parsed from the url
```

```
      url_redirect <- reactive({
        params <- sd_get_url_pars()
        id <- params["id"]
        return(paste0("https://www.google.com?id=", id))
      })

      # Create the redirect button
      sd_redirect(
        id = "redirect_url_pars",
        url = url_redirect(),
        button = TRUE,
        label = "Redirect"
      )

      sd_skip_if(
        input$screening_question == "end_1" ~ "end_page_1",
        input$screening_question == "end_1" ~ "end_page_2",
      )

      sd_server()
    }

    # Run the app
    shiny::shinyApp(ui = sd_ui(), server = server)

    # Clean up
    setwd(orig_dir)
  }
```

sd_server — *Server logic for a surveydown survey*

Description

This function defines the server-side logic for a 'shiny' application used in surveydown. It handles various operations such as conditional display, progress tracking, page navigation, database updates for survey responses, and exit survey functionality.

Usage

```
sd_server(
  db = NULL,
  required_questions = NULL,
  all_questions_required = FALSE,
  start_page = NULL,
  admin_page = FALSE,
  auto_scroll = FALSE,
  rate_survey = FALSE,
  language = "en",
  use_cookies = TRUE
)
```

Arguments

`db`	A list containing database connection information created using `sd_database()` function. Defaults to `NULL`.
`required_questions`	Vector of character strings. The IDs of questions that must be answered. Defaults to `NULL`.
`all_questions_required`	Logical. If `TRUE`, all questions in the survey will be required. Defaults to `FALSE`.
`start_page`	Character string. The ID of the page to start on. Defaults to `NULL`.
`admin_page`	Logical. Whether to include an admin page for viewing and downloading survey data. Defaults to `FALSE`.
`auto_scroll`	Logical. Whether to enable auto-scrolling to the next question after answering. Defaults to `FALSE`.
`rate_survey`	Logical. If `TRUE`, shows a rating question when exiting the survey. If `FALSE`, shows a simple confirmation dialog. Defaults to `FALSE`.
`language`	Set the language for the survey system messages. Include your own in a `translations.yml` file, or choose a built in one from the following list: English (`"en"`), German (`"de"`), Spanish (`"es"`), French (`"fr"`), Italian (`"it"`). Simplified Chinese (`"zh-CN"`). Defaults to `"en"`.
`use_cookies`	Logical. If `TRUE`, enables cookie-based session management for storing and restoring survey progress. Defaults to `TRUE`.

Details

The function performs the following tasks:

- Initializes variables and reactive values.
- Implements conditional display logic for questions.
- Tracks answered questions and updates the progress bar.
- Handles page navigation and skip logic.
- Manages required questions.
- Performs database operation.
- Sets up admin functionality if enabled with the `admin_page` argument.
- Controls auto-scrolling behavior based on the `auto_scroll` argument.
- Uses sweetalert for warning messages when required questions are not answered.
- Handles the exit survey process based on the `rate_survey` argument.

Value

This function does not return a value; it sets up the server-side logic for the 'shiny' application.

Progress Bar

The progress bar is updated based on the last answered question. It will jump to the percentage corresponding to the last answered question and will never decrease, even if earlier questions are answered later. The progress is calculated as the ratio of the last answered question's index to the total number of questions.

Database Operations

If `db` is provided, the function will update the database with survey responses. If `db` is `NULL` (ignore mode), responses will be saved to a local CSV file.

Auto-Scrolling

When `auto_scroll` is `TRUE`, the survey will automatically scroll to the next question after the current question is answered. This behavior can be disabled by setting `auto_scroll = FALSE`.

Exit Survey

When `rate_survey = TRUE`, the function will show a rating question when the user attempts to exit the survey. When `FALSE`, it will show a simple confirmation dialog. The rating, if provided, is saved with the survey data.

See Also

`sd_database()`, `sd_ui()`

Examples

```
if (interactive()) {
  library(surveydown)

  # Get path to example survey file
  survey_path <- system.file("examples", "basic_survey.qmd",
                             package = "surveydown")

  # Copy to a temporary directory
  temp_dir <- tempdir()
  file.copy(survey_path, file.path(temp_dir, "survey.qmd"))
  orig_dir <- getwd()
  setwd(temp_dir)

  # Define a minimal server
  server <- function(input, output, session) {

    # sd_server() accepts these following parameters
    sd_server(
      db = NULL,
      required_questions = NULL,
      all_questions_required = FALSE,
      start_page = NULL,
      admin_page = FALSE,
      auto_scroll = FALSE,
      rate_survey = FALSE,
      language = "en",
      use_cookies = TRUE
    )
  }

  # Run the app
  shiny::shinyApp(ui = sd_ui(), server = server)

  # Clean up
  setwd(orig_dir)
}
```

sd_setup *Required Set Up Function*

Description

This function is depreciated and no longer needed.

Usage

```
sd_setup()
```

Details

The function configures the 'shiny' application to use Bootstrap 5 for styling and enables 'shinyjs' for JavaScript functionalities within the application.

Value

This function does not return a value. It is called for its side effects of setting up the 'shiny' application.

sd_set_password *Set password for surveydown survey*

Description

This function sets your surveydown password, which is used to access the 'PostgreSQL' data (e.g. Supabase). The password is saved in a `.Renviron` file and adds `.Renviron` to `.gitignore`.

Usage

```
sd_set_password(password)
```

Arguments

`password`	Character string. The password to be set for the database connection.

Details

The function performs the following actions:

1. Creates a `.Renviron` file in the root directory if it doesn't exist.
2. Adds or updates the `SURVEYDOWN_PASSWORD` entry in the `.Renviron` file.
3. Adds `.Renviron` to `.gitignore` if it's not already there.

Value

None. The function is called for its side effects.

Examples

```
## Not run:
  # Set a temporary password for demonstration
  temp_password <- paste0(sample(letters, 10, replace = TRUE), collapse = "")

  # Set the password
  sd_set_password(temp_password)

  # After restarting R, verify the password was set
  cat("Password is :", Sys.getenv('SURVEYDOWN_PASSWORD'))

## End(Not run)
```

`sd_show_if` *Define show conditions for survey questions*

Description

This function is used to define conditions under which certain questions in the survey should be shown. It takes one or more formulas where the left-hand side is the condition and the right-hand side is the target question ID. If called with no arguments, it will return `NULL` and set no conditions.

Usage

```
sd_show_if(...)
```

Arguments

`...`	One or more formulas defining show conditions. The left-hand side of each formula should be a condition based on input values, and the right-hand side should be the ID of the question to show if the condition is met.

Value

A list of parsed conditions, where each element contains the condition and the target question ID. Returns `NULL` if no conditions are provided.

See Also

`sd_skip_if()`

Examples

```
if (interactive()) {
  library(surveydown)

  # Get path to example survey file
  survey_path <- system.file("examples", "sd_show_if.qmd",
                             package = "surveydown")

  # Copy to a temporary directory
  temp_dir <- tempdir()
  file.copy(survey_path, file.path(temp_dir, "survey.qmd"))
  orig_dir <- getwd()
  setwd(temp_dir)

  # Define a minimal server
  server <- function(input, output, session) {

    sd_show_if(
      # If "Other" is chosen, show the conditional question
      input$fav_fruit == "other" ~ "fav_fruit_other"
    )

    sd_server()
  }

  # Run the app
  shiny::shinyApp(ui = sd_ui(), server = server)

  # Clean up
  setwd(orig_dir)
}
```

`sd_show_password` *Show the Saved Survey Password*

Description

This function displays the password saved in the `.Renviron` file under the `SURVEYDOWN_PASSWORD` variable. It includes a confirmation step to ensure the user wants to display the password in the console. If no password is found, it suggests using the `sd_set_password()` function to define a password.

Usage

```
sd_show_password()
```

Value

A character string containing the password if found and confirmed, or a message if no password is saved along with a suggestion to set one.

Examples

```
## Not run:
  surveydown::sd_show_password()

## End(Not run)
```

`sd_skip_if` *Define skip conditions for survey pages*

Description

This function is used to define conditions under which certain pages in the survey should be skipped. It takes one or more formulas where the left-hand side is the condition and the right-hand side is the target page ID.

Usage

```
sd_skip_if(...)
```

Arguments

`...`	One or more formulas defining skip conditions. The left-hand side of each formula should be a condition based on input values, and the right-hand side should be the ID of the page to skip to if the condition is met.

Value

A list of parsed conditions, where each element contains the condition and the target page ID.

See Also

`sd_show_if()`

Examples

```
if (interactive()) {
  library(surveydown)

  # Get path to example survey file
  survey_path <- system.file("examples", "sd_skip_if.qmd",
                             package = "surveydown")

  # Copy to a temporary directory
  temp_dir <- tempdir()
  file.copy(survey_path, file.path(temp_dir, "survey.qmd"))
  orig_dir <- getwd()
  setwd(temp_dir)

  # Define a minimal server
  server <- function(input, output, session) {

    # Skip to page based on input
    sd_skip_if(
      input$fav_fruit == "orange" ~ "orange_page",
      input$fav_fruit == "other" ~ "other_page"
    )

    sd_server()
  }

  # Run the app
  shiny::shinyApp(ui = sd_ui(), server = server)

  # Clean up
  setwd(orig_dir)
}
```

`sd_store_value` *Store a value in the survey data*

Description

This function allows storing additional values to be included in the survey data, such as respondent IDs or other metadata.

Usage

```
sd_store_value(value, id = NULL)
```

Arguments

`value`	The value to be stored. This can be any R object that can be coerced to a character string.
`id`	(Optional) Character string. The id (name) of the value in the data. If not provided, the name of the `value` variable will be used.

Value

`NULL` (invisibly)

Examples

```
if (interactive()) {
  library(surveydown)

  # Get path to example survey file
  survey_path <- system.file("examples", "sd_ui.qmd",
                             package = "surveydown")

  # Copy to a temporary directory
  temp_dir <- tempdir()
  file.copy(survey_path, file.path(temp_dir, "basic_survey.qmd"))
  orig_dir <- getwd()
  setwd(temp_dir)

  # Define a minimal server
  server <- function(input, output, session) {

    # Create a respondent ID to store
    respondentID <- 42

    # Store the respondentID
    sd_store_value(respondentID)

    # Store the respondentID as the variable "respID"
    sd_store_value(respondentID, "respID")

    sd_server()
  }

  # Run the app
  shiny::shinyApp(ui = sd_ui(), server = server)

  # Clean up
  setwd(orig_dir)
}
```

`sd_ui` *Create the UI for a surveydown survey*

Description

This function creates the user interface for a surveydown survey, including necessary CSS and JavaScript files, and applies custom styling. It retrieves theme and progress bar settings from the survey.qmd file.

Usage

```
sd_ui()
```

Details

The function reads the following settings from the survey.qmd YAML header:

- `theme`: The theme to be applied to the survey.
- `barcolor`: The color of the progress bar (should be a valid hex color).
- `barposition`: The position of the progress bar (`'top'`, `'bottom'`, or `'none'`).

If `barcolor` is not specified or is `NULL`, the default theme color will be used. If `barposition` is not specified, it defaults to 'top'.

Value

A 'shiny' UI object

See Also

`sd_server()` for creating the server-side logic of the survey

Examples

```
if (interactive()) {
  library(surveydown)

  # Get path to example survey file
  survey_path <- system.file("examples", "sd_ui.qmd",
                             package = "surveydown")

  # Copy to a temporary directory
  temp_dir <- tempdir()
  file.copy(survey_path, file.path(temp_dir, "survey.qmd"))
  orig_dir <- getwd()
  setwd(temp_dir)

  # Define a minimal server
  server <- function(input, output, session) {
    sd_server()
  }

  # Run the app
  shiny::shinyApp(ui = sd_ui(), server = server)

  # Clean up
  setwd(orig_dir)
}
```

`sd_version` *Check Surveydown Version*

Description

This function checks if the local surveydown package is up-to-date with the latest online version. It compares the local version with the latest version available on GitHub and provides information about whether an update is needed.

Usage

```
sd_version()
```

Value

No return value, called for side effects (prints version information and update status to the console).

Examples

```
surveydown::sd_version()
```

Index