

Package ‘surveydown’

August 20, 2025

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

Version 0.13.0

Description Generate programmable 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

URL <https://pkg.surveydown.org>

BugReports <https://github.com/surveydown-dev/surveydown/issues>

Depends R (>= 4.1.0)

Imports cli, DBI, dotenv, fs, htmltools, jsonlite, markdown, miniUI, pool, quarto, RPostgres, rstudioapi, rvest, shiny, shinyjs, shinyWidgets, utils, xml2, yaml

Suggests glue, knitr, leaflet, testthat

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

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

<<https://orcid.org/0009-0006-2942-0588>>),
Stefan Munnes [ctb]

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

Repository CRAN

Date/Publication 2025-08-20 20:30:07 UTC

Contents

sd_add_page	3
sd_add_question	4
sd_close	5
sd_completion_code	7
sd_copy_value	8
sd_create_survey	9
sd_create_translations	10
sd_dashboard	11
sd_database	12
sd_db_config	13
sd_db_connect	15
sd_display_question	16
sd_display_value	16
sd_get_data	17
sd_get_url_pars	18
sd_include_folder	19
sd_is_answered	20
sd_next	21
sd_output	22
sd_page_gadget	24
sd_question	24
sd_question_custom	27
sd_question_gadget	29
sd_reactive	29
sd_redirect	30
sd_server	32
sd_setup	35
sd_set_password	36
sd_show_if	37
sd_skip_forward	38
sd_skip_if	39
sd_store_value	40
sd_ui	41
sd_version	43

Index	44
--------------	-----------

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(page_id = "page_id")
```

Arguments

`page_id` A character string specifying the ID for the page. Defaults to "page_id".

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 the specified 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

Special `page_id` values:

- When `page_id` is "end", a thank-you page template with `sd_close()` is inserted

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 with default ID
  sd_add_page()

  # Insert a new page template with custom ID
  sd_add_page(page_id = "welcome")

  # Insert an end/thank you page
  sd_add_page(page_id = "end")
}
```

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", id = NULL, label = NULL, 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
------	--

id	A character string specifying the ID for the question. If not provided, a default ID based on the question type will be used. This ID should be unique within your survey.
label	A character string specifying the label (question text) to display to respondents. If not provided, a default label placeholder will be used.
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.
3. If an ID is provided, replaces the default ID in the template with the provided ID.
4. If a label is provided, replaces the default label in the template with the provided label.

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 with custom ID and label
  sd_add_question("text", id = "user_email", label = "What is your email address?")

  # Insert a slider question template
  sd_add_question("slider", id = "satisfaction", label = "How satisfied were you with our service?")
}
```

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	<i>Generate a Random Completion Code</i>
--------------------	--

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	<i>Create a copy of a value</i>
---------------	---------------------------------

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

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

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 template files to the specified directory. You can choose from various predefined templates, including the default built-in template and specialized templates from the surveydown-dev/templates repository.

Usage

```
sd_create_survey(template = "default", path = getwd(), ask = TRUE)
```

Arguments

template	A character string specifying the template to use. Default is "default" which uses the built-in package template. Other options include: default The default built-in template conditional_display Template of conditional display of questions conditional_navigation Template of conditional navigation of pages conjoint_buttons Conjoint analysis with button interface conjoint_tables Conjoint analysis with table interface custom_leaflet_map Survey with interactive Leaflet maps custom_plotly_chart Survey with Plotly visualizations external_redirect Template with external site redirects live_polling Live polling template for real-time surveys question_types Showcases all available question types questions_yaml Survey with questions defined in a YAML file random_options Survey with randomized question options random_options_predefined Randomized options from predefined sets reactive_drilldown Dynamic questions with drill-down capability reactive_questions Survey with reactive questions
path	A character string specifying the directory where the survey template should be created. Defaults to the current working directory.
ask	Logical. If TRUE (default), prompts for user confirmation when creating the survey in the current directory. If FALSE, bypasses the confirmation prompt and proceeds without asking.

Details

When creating a new survey template, this function will:

1. Check if the specified template is valid
2. Confirm the destination path with the user (if it's the current directory)
3. Download template files from GitHub if a non-default template is specified
4. Copy template files to the destination directory
5. Skip .Rproj files if one already exists in the destination
6. Prompt for confirmation before overwriting existing files

External templates are downloaded from the `surveydown-dev/templates` GitHub repository.

Value

Invisible NULL. The function is called for its side effects.

Examples

```
if (interactive()) {
  # Create a survey with the "question_types" template in the "my_survey" directory
  sd_create_survey(template = "question_types", path = "my_survey")

  # Create a survey using the default template in the "my_survey" directory
  sd_create_survey(path = "my_survey")

  # Create a survey with default template in current directory
  sd_create_survey("default")

  # Create a survey without asking for confirmation
  sd_create_survey(template = "default", path = "my_survey", ask = FALSE)
}
```

`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/dateinput for supported languages. Also, if "en", "de", "es", "fr", or "it" is chosen, default messages in those languages 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_dashboard

Deprecated Survey Dashboard

Description

This dashboard was depreciated in version v0.13.0. Now the sdstudio package fully includes the functionality that was previously included in this function.

Usage

```
sd_dashboard(gssencmode = "auto")
```

Arguments

gssencmode	Character string. The GSS encryption mode for the database connection. Defaults to "auto". Options are: "auto": Tries "prefer" first, then falls back to "disable" if GSSAPI negotiation fails
------------	--

- "prefer": Uses GSSAPI encryption if available, plain connection otherwise
- "disable": Disables GSSAPI encryption entirely Set to "disable" if you're having connection issues on a secure connection like a VPN.

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 PostgreSQL database.
dbname	Character string. The name of the PostgreSQL database.
port	Integer. The port number for the PostgreSQL database connection.
user	Character string. The username for the PostgreSQL database connection.
table	Character string. The name of the table to interact with in the Supabase database.
password	Character string. The password for the PostgreSQL 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 <code>surveydown::sd_set_password()</code> , which will create a local <code>.Renv</code> file that stores your password as a <code>SURVEYDOWN_PASSWORD</code> environment variable. The password argument uses this as the default value, so if you set a password properly with <code>surveydown::sd_set_password()</code> , 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   = "6443",
    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_db_config	Configure database settings
--------------	-----------------------------

Description

Set up or modify database configuration settings in a .env file. These settings are used to establish database connections for storing survey responses.

Usage

```
sd_db_config(  
  host = NULL,  
  dbname = NULL,  
  port = NULL,  
  user = NULL,  
  table = NULL,  
  password = NULL,  
  interactive = NULL  
)
```

Arguments

host	Character string. Database host
dbname	Character string. Database name
port	Character string. Database port
user	Character string. Database user
table	Character string. Table name
password	Character string. Database password
interactive	Logical. Whether to use interactive setup. Defaults to TRUE if no parameters provided

Value

Invisibly returns a list of the current configuration settings

See Also

- [sd_db_connect\(\)](#) to connect to the database

Examples

```
if (interactive()) {  
  # Interactive setup  
  sd_db_config()  
  
  # Update specific settings  
  sd_db_config(table = "new_table")  
  
  # Update multiple settings  
  sd_db_config(  
    host = "new_host",  
    port = "5433",  
    table = "new_table"  
  )  
}
```

sd_db_connect	<i>Connect to database</i>
---------------	----------------------------

Description

Establish a connection to the database using settings from .env file. This function creates a connection pool for efficient database access and provides options for local data storage when needed.

Usage

```
sd_db_connect(env_file = ".env", ignore = FALSE, gssencmode = "auto")
```

Arguments

- | | |
|------------|--|
| env_file | Character string. Path to the env file. Defaults to ".env" |
| ignore | Logical. If TRUE, data will be saved to a local CSV file instead of the database. Defaults to FALSE. |
| gssencmode | Character string. The GSS encryption mode for the database connection. Defaults to "auto". Options are: <ul style="list-style-type: none"> "auto": Tries "prefer" first, then falls back to "disable" if GSSAPI negotiation fails "prefer": Uses GSSAPI encryption if available, plain connection otherwise "disable": Disables GSSAPI encryption entirely NOTE: If you have verified all connection details are correct but still cannot access the database, try setting this to "disable". |

Value

A list containing the database connection pool (db) and table name (table), or NULL if ignore is TRUE or if connection fails

Examples

```
if (interactive()) {
  # Connect using settings from .env
  db <- sd_db_connect()

  # Use local storage instead of database
  db <- sd_db_connect(ignore = TRUE)

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

sd_display_question	<i>Create a placeholder for a reactive survey question</i>
---------------------	--

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	<i>Display the value of a survey question</i>
------------------	---

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, table = NULL, refresh_interval = NULL)
```

Arguments

db	A list containing database connection details created using <code>sd_db_config()</code> . Must have elements: • db: A DBI database connection object • table: A string specifying the name of the table to query
table	Character string. Database table name to obtain data from, overrides the table provided in the db argument. Defaults to NULL.
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	<i>Include a folder to the 'shiny' resource path</i>
-------------------	--

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	<i>Check if a question is answered</i>
----------------	--

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

<code>id</code>	Character string. A unique identifier for the output element.
<code>type</code>	Character string. Specifies the type of output corresponding with the question id. Can be "question", "value", "label_option", "label_question", or NULL. If "question", it will display a question defined in the server. If "value", it will display the value of question id selected by the respondent. If "label_option", it will display the label of the option for question id selected by the respondent. If "label_question", it will display the label argument value for question id. Finally, if NULL, the function behaves like <code>shiny::uiOutput()</code> .
<code>width</code>	Character string. The width of the UI element. Defaults to "100%".
<code>display</code>	Character string. Specifies the display type for "value" outputs. Can be "text", "verbatim", or "ui". Only used when <code>type = "value"</code> .
<code>inline</code>	Logical. Whether to render the output inline. Defaults to TRUE.
<code>wrapper</code>	Function. A function to wrap the output. Only used when <code>type = "value"</code> .
<code>...</code>	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_page_gadget

Show a Shiny gadget for entering a page ID

Description

This function displays a Shiny gadget that allows the user to input a page ID. Once submitted, it calls `sd_add_page()` with the specified ID.

Usage

```
sd_page_gadget()
```

Value

The entered page ID (invisibly).

sd_question

Create a survey question

Description

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

Usage

```

sd_question(
  id,
  type = NULL,
  label = NULL,
  option = NULL,
  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,
    placeholder = NULL,
    resize = NULL,
    row = NULL,
    default = NULL,
    yml = "questions.yml",
    ...
)

```

Arguments

id	A unique identifier for the question, which will be used as the variable name in the resulting survey data.
type	Specifies the type of question. Possible values are "select", "mc", "mc_multiple", "mc_buttons", "mc_multiple_buttons", "text", "textarea", "numeric", "slider", "slider_numeric", "date", "daterange", and "matrix". Defaults to NULL.
label	Character string. The label for the UI element, which can be formatted with markdown. Defaults to NULL.
option	Named vector for the "select", "radio", "checkbox", and "slider" question types, or numeric vector for "slider_numeric" question type.
cols	Integer. Number of columns for the "textarea" question type. 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 input for the "textarea" question type. 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.
placeholder	Character string. Placeholder text for "text" and "textarea" question types.
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.

default	Numeric, length 1 (for a single sided slider), or 2 for a two sided (range based) slider. Values to be used as the starting default for the slider. Defaults to the median of values.
yaml	Character string. The name of the YAML file to load question configurations from. Defaults to "questions.yaml". Custom YAML files can be specified, either in the root directory or subdirectories (e.g., "folder/custom.yaml").
...	Additional arguments, often specific to different input types. Examples include pre, sep, step, and animate for "slider" and "slider_numeric" question types, etc.

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 question
- "textarea": Multi-line text question
- "numeric": Numeric question
- "slider": Slider question
- "slider_numeric": Extended numeric slider question
- "date": Date question
- "daterange": Date range question
- "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., <code>leafletOutput()</code> , <code>plotlyOutput()</code>).
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_question_gadget	<i>Show a Shiny gadget for selecting a question type</i>
--------------------	--

Description

This function displays a Shiny gadget that allows the user to select a question type from a dropdown menu. Once submitted, it calls `sd_add_question()` with the specified type.

Usage

```
sd_question_gadget(chunk = FALSE)
```

Arguments

chunk	Logical. If TRUE, the code will be generated with the R code chunk wrapper. Defaults to FALSE.
-------	--

Value

The selected question type (invisibly).

sd_reactive	<i>Create a reactive value that is also stored in survey data</i>
-------------	---

Description

This function creates a reactive value similar to Shiny's `reactive()` function, but also automatically stores the calculated value in the survey data.

Usage

```
sd_reactive(id, expr, blank_na = TRUE)
```

Arguments

id	Character string. The id (name) of the value to be stored in the data.
expr	An expression that calculates a value based on inputs
blank_na	Logical. If TRUE, NA values are converted to empty strings. Default is TRUE.

Value

A reactive expression that can be called like a function

Examples

```
# This example shows how sd_reactive would be used in the app.R file
if (interactive()) {
  library(surveydown)
  library(shiny)

  # Demo app setup
  server <- function(input, output, session) {
    # Create a reactive value that is stored in survey data
    product <- sd_reactive("product", {
      as.numeric(input$first_number) * as.numeric(input$second_number)
    })

    # Display the result
    output$result <- renderText({
      paste("The product is:", product())
    })

    # The rest of your survey setup...
    sd_server()
  }

  # In your survey.qmd file, you would use:
  # The product is: `r sd_output("product", type = "value")`
}
```

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. In reactive contexts, this becomes the output ID, while the actual button gets the ID id + "_btn" to avoid input/output conflicts.
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, creates an output with the specified ID that can be displayed using `sd_output()`. The actual button element gets the ID id + "_btn" to prevent input/output conflicts. 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,
  auto_scroll = FALSE,
  rate_survey = FALSE,
  system_language = "en",
  use_cookies = TRUE,
  highlight_unanswered = TRUE,
  highlight_color = "gray",
  capture_metadata = TRUE,
  language = NULL
)

```


Arguments

<code>db</code>	A list containing database connection information created using <code>sd_database()</code> function. Defaults to <code>NULL</code> . If <code>NULL</code> , will be auto-detected from the calling environment or remain <code>NULL</code> (ignore mode).
<code>required_questions</code>	Vector of character strings. The IDs of questions that must be answered. Defaults to <code>NULL</code> (no required questions).
<code>all_questions_required</code>	Logical. If <code>TRUE</code> , all questions in the survey will be required. Defaults to <code>FALSE</code> .
<code>start_page</code>	Character string. The ID of the page to start on. Defaults to <code>NULL</code> (first page).
<code>auto_scroll</code>	Logical. Whether to enable auto-scrolling to the next question after answering. Defaults to <code>FALSE</code> .
<code>rate_survey</code>	Logical. If <code>TRUE</code> , shows a rating question when exiting the survey. If <code>FALSE</code> , shows a simple confirmation dialog. Defaults to <code>FALSE</code> .
<code>system_language</code>	Set the language for the survey system messages. Include your own in a <code>translations.yml</code> file, or choose a built in one from the following list: English (" <code>en</code> "), German (" <code>de</code> "), Spanish (" <code>es</code> "), French (" <code>fr</code> "), Italian (" <code>it</code> "), Simplified Chinese (" <code>zh-CN</code> "). Defaults to " <code>en</code> ". Note: The deprecated language parameter is still supported for backward compatibility.
<code>use_cookies</code>	Logical. If <code>TRUE</code> , enables cookie-based session management for storing and restoring survey progress. Defaults to <code>TRUE</code> . Can be overridden by <code>use_cookies</code> setting in the survey.qmd YAML header.
<code>highlight_unanswered</code>	Logical. If <code>TRUE</code> , enables highlighting of all unanswered questions on page display. Defaults to <code>TRUE</code> .
<code>highlight_color</code>	Character string. Color for highlighting unanswered questions. Options are " <code>blue</code> ", " <code>orange</code> ", " <code>green</code> ", " <code>purple</code> ", " <code>gray</code> ", or " <code>grey</code> ". Defaults to " <code>gray</code> ".
<code>capture_metadata</code>	Logical. If <code>TRUE</code> , automatically captures and stores browser information (browser name, version, and OS), IP address, and screen resolution. Defaults to <code>TRUE</code> .
<code>language</code>	Deprecated as of v0.13.0. Use <code>system_language</code> instead. This parameter. is maintained for backward compatibility only.

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.
- 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 = NULL,
    start_page = NULL,
    auto_scroll = NULL,
    rate_survey = NULL,
    system_language = "en",
    use_cookies = NULL,
    highlight_unanswered = NULL,
    highlight_color = NULL,
    capture_metadata = NULL
  )
}

# 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	<i>Set password for surveydown survey</i>
-----------------	---

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 and pages***Description**

This function is used to define conditions under which certain questions or pages 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 or page 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 or page to show if the condition is met.

Value

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

See Also

```
sd_skip_forward()
```

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$fav_fruit == "other" ~ "fav_fruit_other",
    # If condition is met, show specific page
    input$category == "advanced" ~ "advanced_page"
  )

  sd_server()
}

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

# Clean up
setwd(orig_dir)
}

```

sd_skip_forward

Define forward skip conditions for survey pages

Description

This function is used to define conditions under which certain pages in the survey should be skipped ahead to (forward only). 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_forward(...)
```

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. Only forward skipping (to pages later in the sequence) is allowed.

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_forward.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 forward to specific pages based on fruit selection
    sd_skip_forward(
      input$fav_fruit == "apple" ~ "apple_page",
      input$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_skip_if

*Define skip conditions for survey pages (Deprecated)***Description**

This function is deprecated. Please use `sd_skip_forward()` instead.

This function is used to define conditions under which certain pages in the survey should be skipped. It now behaves like `sd_skip_forward()` where only forward skipping is allowed to prevent navigation loops.

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.

sd_store_value	<i>Store a value in the survey data</i>
----------------	---

Description

This function allows storing additional values to be included in the survey data, such as respondent IDs or other metadata. When a database connection is provided, it implements session persistence - if a value already exists for the current session, storage is skipped to maintain consistency across page refreshes.

Usage

```
sd_store_value(value, id = NULL, db = NULL, auto_assign = TRUE)
```

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.
db	(Optional) Database connection object created with <code>sd_db_connect()</code> . If provided, enables session persistence. If not provided, will automatically look for a variable named 'db' in the calling environment, or fall back to the database connection from the session.
auto_assign	Logical. If TRUE (default), automatically assigns the stored value back to the original variable in the calling environment. This eliminates the need for explicit assignment when session persistence is desired. If FALSE, the function only returns the value without modifying the original variable.

Value

The value that was stored (either the new value or existing value from database if session persistence applies). This allows the function to be used in variable assignments.

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) {
    # Set up database connection
    db <- sd_db_connect()

    # Generate and store values with automatic assignment (default behavior)
    respondentID <- sample(1:1000, 1)
    sd_store_value(respondentID, "respID", db) # respondentID automatically updated

    completion_code <- sample(0:9, 6, replace = TRUE)
    sd_store_value(completion_code) # completion_code automatically updated

    # Traditional assignment approach (auto_assign = FALSE)
    some_value <- sd_store_value(42, "some_value", auto_assign = FALSE)

    # The function ensures session persistence across page refreshes

    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

* database functions

sd_db_config, [13](#)

sd_add_page, [3](#)

sd_add_question, [4](#)

sd_close, [5](#)

sd_completion_code, [7](#)

sd_copy_value, [8](#)

sd_create_survey, [9](#)

sd_create_translations, [10](#)

sd_dashboard, [11](#)

sd_database, [12](#)

sd_db_config, [13](#)

sd_db_connect, [15](#)

sd_db_connect(), [14](#)

sd_display_question, [16](#)

sd_display_value, [16](#)

sd_get_data, [17](#)

sd_get_url_pars, [18](#)

sd_include_folder, [19](#)

sd_is_answered, [20](#)

sd_next, [21](#)

sd_output, [22](#)

sd_page_gadget, [24](#)

sd_question, [24](#)

sd_question(), [28](#)

sd_question_custom, [27](#)

sd_question_gadget, [29](#)

sd_reactive, [29](#)

sd_redirect, [30](#)

sd_server, [6](#), [32](#)

sd_set_password, [36](#)

sd_setup, [35](#)

sd_show_if, [37](#)

sd_skip_forward, [38](#)

sd_skip_if, [39](#)

sd_store_value, [40](#)

sd_ui, [41](#)

sd_version, [43](#)