Package ‘shinysurveys’

July 11, 2021

Title Create and Deploy Surveys in 'Shiny'

Version 0.2.0

Description Easily create and deploy surveys in 'Shiny'. This package includes
a minimalistic framework similar to 'Google Forms' that allows for url-based
user tracking, customizable submit actions, easy survey-theming, and more.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1.9000

Imports shiny, sass, htmltools, jsonlite

Depends R (>= 2.10)

Suggests knitr, rmarkdown, tibble

VignetteBuilder knitr

NeedsCompilation no

Author Jonathan Trattner [aut, cre] (<`https://orcid.org/0000-0002-1097-7603`>),
Lucy D'Agostino McGowan [aut],
Paul Le Grand [ctb]

Maintainer Jonathan Trattner <`jdt@jdtrat.com`>

Repository CRAN

Date/Publication 2021-07-11 04:10:02 UTC

R topics documented:

`demo_survey` *Demo Survey*

Description

This function runs a Shiny app that shows an example of running a demographic survey in Shiny. It has a sample title and description and its theme color can be customized using a hex color code.

Usage

```
demo_survey(theme = "#63B8FF")
```

Arguments

`theme`	A valid hex color such as #63B8FF (default)

Value

A Shiny App

Examples

```
if (interactive()) demo_survey()
```

`demo_survey_multipage` *Demo Survey over Multiple Pages*

Description

This function runs a Shiny app that shows an example of running a demographic survey in Shiny, spanning multiple pages. It has a sample title and description and its theme color can be customized using a hex color code.

Usage

```
demo_survey_multipage(theme = "#63B8FF")
```

Arguments

`theme`	A valid hex color such as #63B8FF (default)

Value

A Shiny App

Examples

```
if (interactive()) demo_survey_multipage()
```

`extendInputType` *Add Custom Input Types for a Survey*

Description

Add Custom Input Types for a Survey

Usage

```
extendInputType(input_type, extension)
```

Arguments

`input_type`	A string of the input type supplied in the data frame of questions.
`extension`	A shiny input type not natively supported by shinysurveys. See the examples section for more information.

Value

NA; used to register custom input types for use with a shiny survey.

See Also

`surveyID`

`surveyLabel`

Examples

```

# Register a slider input to {shinysurveys} with a custom minimum and maximum value.

extendInputType("slider", {
  shiny::sliderInput(
    inputId = surveyID(),
    label = surveyLabel(),
    min = 1,
    max = 10,
    value = 5
    )
  })

# Define a question as normal with the `input_type` set to the custom slider type defined above.
slider_question <- data.frame(question = "On a scale from 1-10,
how much do you love sushi?",
option = NA,
input_type = "slider",
input_id = "sushi_scale",
dependence = NA,
dependence_value = NA,
required = TRUE)

# Watch it in action
if (interactive()) {
ui <- fluidPage(
  surveyOutput(df = slider_question, "Sushi Scale Example")
)

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

shinyApp(ui, server)

}



# Register a date input to {shinysurveys},
# limiting possible dates to a twenty-day period.

extendInputType("date", {
  shiny::dateInput(
    inputId = surveyID(),
    value = Sys.Date(),
    label = surveyLabel(),
    min = Sys.Date()-10,
    max = Sys.Date()+10
  )
})

# Define a question as normal with the `input_type` set to
# the custom date type defined above.

date_question <- data.frame(question = "When do you graduate?",
option = NA,
input_type = "date",
input_id = "grad_date",
dependence = NA,
dependence_value = NA,
required = FALSE)

# Watch it in action
if (interactive()) {
ui <- fluidPage(
  surveyOutput(df = date_question, "Date Input Extension Example")
)

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

shinyApp(ui, server)
}


# Combine both custom input types:

if (interactive()) {
ui <- fluidPage(
  surveyOutput(df = rbind(slider_question, date_question),
  "Date & Slider Input Extension Example")
)

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

shinyApp(ui, server)
}
```

getSurveyData *Get survey data*

Description

Get a participant's responses.

Usage

```
getSurveyData(
  custom_id = NULL,
  include_dependencies = TRUE,
  dependency_string = "HIDDEN-QUESTION"
)
```

Arguments

`custom_id`	A unique identifier for the survey's respondents. NULL by default, and the built-in shinysurveys userID will be used.
`include_dependencies`	LOGICAL: TRUE (default) and all dependency questions will be returned, regardless of if the individual respondent saw it. For respondents who did not see a specific question, the 'response' will take on the value from the `dependency_string` argument. If FALSE, the output will have variable rows depending on which questions a given participant answered.
`dependency_string`	A character string to be imputed for dependency questions that a respondent did not see. Default is "HIDDEN-QUESTION".

Value

A data frame with four columns containing information about the participant's survey responses: The 'subject_id' column can be used for identifying respondents. By default, it utilizes shinysurveys URL-based user tracking feature. The 'question_id' and 'question_type' columns correspond to 'input_id' and 'input_type' from the original data frame of questions. The 'response' column is the participant's answer.

The number of rows, corresponding to the questions an individual saw, depends on the `include_dependencies` argument. If TRUE, by default, then the resulting data frame will have one row per unique input ID. If FALSE, the data frame may have variable length depending on which questions a given individual answers.

Examples

```
if (interactive()) {

 library(shiny)

 ui <- fluidPage(
   surveyOutput(teaching_r_questions)
 )

 server <- function(input, output, session) {
   renderSurvey()
   # Upon submission, print a data frame with participant responses
   observeEvent(input$submit, {
     print(getSurveyData())
   })
 }

 shinyApp(ui, server)

 }
```

`listInputExtensions` *List all registered survey extensions*

Description

List all registered survey extensions

Usage

```
listInputExtensions()
```

Value

A named list containing the registered input type and their associated functions.

Examples

```
if (interactive()) {

  # Register a date input to {shinysurveys},
  # limiting possible dates to a twenty-day period.

  extendInputType("slider", {
    shiny::sliderInput(
      inputId = surveyID(),
      label = surveyLabel(),
      min = 1,
      max = 10,
      value = 5
      )
    })

  # Register a slider input to {shinysurveys}
  # with a custom minimum and maximum value.

  extendInputType("date", {
    shiny::dateInput(
      inputId = surveyID(),
      value = Sys.Date(),
      label = surveyLabel(),
      min = Sys.Date()-10,
      max = Sys.Date()+10
    )
  })

  listInputExtensions()

}
```

`numberInput` *Create a numeric input*

Description

Create an input control for entry of numeric values. This is identical to `shiny::numericInput()` but is more flexible in **not** requiring an initial value and in allowing placeholders.

Usage

```
numberInput(
  inputId,
  label,
  value = NULL,
  min = NA,
  max = NA,
  step = NA,
  placeholder = NULL,
  width = NULL
)
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`label`	Display label for the control, or `NULL` for no label.
`value`	Initial value. NULL by default.
`min`	Minimum allowed value
`max`	Maximum allowed value
`step`	Interval to use when stepping between min and max
`placeholder`	A character string giving the user a hint as to what can be entered into the control. Internet Explorer 8 and 9 do not support this option.
`width`	The width of the input, e.g. `'400px'`, or `'100%'`; see `validateCssUnit()`.

Value

A numeric input control that can be added to a UI definition.

Server value

A numeric vector of length 1.

See Also

`shiny::updateNumericInput()`

Examples

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

ui <- fluidPage(
  numberInput("obs", "Observations:", placeholder = "How many do you see?", min = 1, max = 100),
   verbatimTextOutput("value")
)
server <- function(input, output) {
   output$value <- renderText({ input$obs })
}
shinyApp(ui, server)
}
```

`radioMatrixInput` *Create a matrix of radio buttons.*

Description

Create a matrix of radio buttons.

Usage

```
radioMatrixInput(inputId, responseItems, choices, selected = NULL, ...)
```

Arguments

`inputId`	The input id
`responseItems`	The questions to be asked (row labels)
`choices`	Possible choices (column labels)
`selected`	Initial selected value
`...`	Additional arguments specific to shinysurveys required questions.

Value

A matrix of radio buttons that can be added to a UI definition. When run in a Shiny application, this will return `NULL` until all possible response items have been answered, at which time a data frame with the question_id, question_type, and response, the format used in `getSurveyData`.

Examples

```
# For use as a normal Shiny input:

if (interactive()) {

  library(shiny)

  ui <- fluidPage(
    radioMatrixInput("matInput",
                     responseItems = c("Love sushi?", "Love chocolate?"),
                     choices = c("Disagree", "Neutral", "Agree"))
  )

  server <- function(input, output, session) {
    observe({
      print(input$matInput)
    })
  }

  shinyApp(ui, server)

}

# For use in {shinysurveys}

if (interactive()) {

 df <- data.frame(
   question = c(rep("I love sushi.", 3), rep("I love chocolate.",3),
   "What's your favorite food?", rep("Goat cheese is the GOAT.", 5),
   rep("Yogurt and berries are a great snack.",5),
   rep("SunButter® is a fantastic alternative to peanut butter.", 5)),
   option = c(rep(c("Disagree", "Neutral", "Agree"), 2), "text",
   rep(c("Strongly Disagree", "Disagree", "Neutral", "Agree", "Strongly Agree"), 3)),
   input_type = c(rep("matrix", 6), "text", rep("matrix", 15)),
   # For matrix questions, the IDs should be the same for each question
   # but different for each matrix input unit
   input_id = c(rep("matId", 6), "favorite_food", rep("matId2", 15)),
   dependence = NA,
   dependence_value = NA,
   required = FALSE
 )

 library(shiny)

 ui <- fluidPage(
   surveyOutput(df)
 )

 server <- function(input, output, session) {
   renderSurvey()
   observe({
     print(input$matId)
     print(input$favorite_food)
     print(input$matId2)
   })
 }

 shinyApp(ui, server)

}
```

`renderSurvey` *Server code for adding survey questions*

Description

Include server-side logic for shinysurveys.

Usage

```
renderSurvey(df, theme = "#63B8FF")
```

Arguments

`df`	**Deprecated** *please only place argument in* `surveyOutput`. A user supplied data frame in the format of teaching_r_questions.
`theme`	**Deprecated** *please place the theme argument in* `surveyOutput`. A valid R color: predefined such as "red" or "blue"; hex colors such as #63B8FF (default). To customize the survey's appearance entirely, supply NULL.

Value

NA; used for server-side logic in Shiny apps.

Examples

```

if (interactive()) {

  library(shiny)
  library(shinysurveys)

  df <- data.frame(question = "What is your favorite food?",
                   option = "Your Answer",
                   input_type = "text",
                   input_id = "favorite_food",
                   dependence = NA,
                   dependence_value = NA,
                   required = F)
  ui <- fluidPage(
    surveyOutput(df = df,
                 survey_title = "Hello, World!",
                 theme = "#63B8FF")
  )

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

    observeEvent(input$submit, {
      showModal(modalDialog(
        title = "Congrats, you completed your first shinysurvey!",
       "You can customize what actions happen when a user finishes a survey using input$submit."
      ))
    })
  }

  shinyApp(ui, server)

}
```

`surveyID` *Add correct ID for custom input types*

Description

`surveyID()` is a helper function for `extendInputType`. When defining custom input types, the `inputId` argument for shiny UI components should equal `surveyID()`. See examples for more details.

Usage

```
surveyID()
```

Value

NA; used for side effects with `extendInputType`.

See Also

`extendInputType`

`surveyLabel`

`surveyOptions`

Examples

```
extendInputType("slider", {
shiny::sliderInput(
  inputId = surveyID(),
  label = surveyLabel(),
  min = 1,
  max = 10,
  value = 5
)
})
```

---

surveyLabel — *Add correct label for custom input types*

Description

`surveyLabel()` is a helper function for `extendInputType`. When defining custom input types, the `label` argument for shiny UI components should equal `surveyLabel()`. It essentially takes on the value in the "question" column in the data supplied to `surveyOutput`. See examples for more details.

Usage

```
surveyLabel()
```

Value

NA; used for side effects with `extendInputType`.

See Also

`extendInputType`

`surveyID`

`surveyOptions`

Examples

```
extendInputType("slider", {
shiny::sliderInput(
  inputId = surveyID(),
  label = surveyLabel(),
  min = 1,
  max = 10,
  value = 5
)
})
```

`surveyOptions` *Add options for custom input types*

Description

`surveyOptions()` is a helper function for `extendInputType`. When defining custom input types, the choices arguments for shiny UI components should equal `surveyOption()`. See examples for more details.

Usage

```
surveyOptions()
```

Value

NA; used for side effects with `extendInputType`.

See Also

`extendInputType`

`surveyID`

`surveyOptions`

Examples

```
extendInputType("inlineRadioButtons", {
shiny::radioButtons(
  inputId = surveyID(),
  label = surveyLabel(),
  selected = character(0),
  choices = surveyOptions(),
  inline = TRUE
)
})
```

surveyOutput *Generate the UI Code for demographic questions*

Description

Create the UI code for a Shiny app based on user-supplied questions.

Usage

```
surveyOutput(df, survey_title, survey_description, theme = "#63B8FF", ...)
```

Arguments

`df`	A user supplied data frame in the format of teaching_r_questions.
`survey_title`	(Optional) user supplied title for the survey
`survey_description`	(Optional) user supplied description for the survey
`theme`	A valid R color: predefined such as "red" or "blue"; hex colors such as #63B8FF (default). To customize the survey's appearance entirely, supply NULL.
`...`	Additional arguments to pass into actionButton used to submit survey responses.

Value

UI Code for a Shiny App.

Examples

```
if (interactive()) {

  library(shiny)
  library(shinysurveys)

  df <- data.frame(question = "What is your favorite food?",
                   option = "Your Answer",
                   input_type = "text",
                   input_id = "favorite_food",
                   dependence = NA,
                   dependence_value = NA,
                   required = F)

  ui <- fluidPage(
    surveyOutput(df = df,
                 survey_title = "Hello, World!",
                 theme = "#63B8FF")
  )

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

    observeEvent(input$submit, {
      showModal(modalDialog(
        title = "Congrats, you completed your first shinysurvey!",
      "You can customize what actions happen when a user finishes a survey using input$submit."
      ))
    })
  }

  shinyApp(ui, server)

}
```

`teaching_r_questions` *A sample CSV file for demographic questions*

Description

A dataset containing the prices and other attributes of almost 54,000 diamonds.

Usage

```
teaching_r_questions
```

Format

A data frame with 54 rows and 6 columns:

question: The question to be asked.

option: A possible response to the question. In multiple choice questions, for example, this would be the possible answers. For questions without discrete answers, such as a numeric input, this would be the default option shown on the input. For text inputs, it is the placeholder value.

input_type: What type of response is expected? Numeric, multiple choice, text, etc...

input_id: The input id for Shiny inputs.

dependence: Does this question (row) depend on another? That is, should it only appear if a different question has a specific value? This column contains the input_id of whatever question this one depends upon.

dependence_value: This column contains the specific value that the dependence question must take for this question (row) to be shown.

required: logical TRUE/FALSE signifying if a question is required.

Source

D'Agostino McGowan Data Science Lab at Wake Forest University.

Index