Building Web Applications in R with Shiny: Case Studies

Ken Harmon

2019 November 15

https://www.datacamp.com/courses/building-web-applications-in-r-with-shiny-case-studies

Google Mapping API

https://cloud.google.com/maps-platform/

Shiny Review

Simple text Every Shiny app has a UI (User Interface) portion and a server portion. The UI is where the visual elements are placed—it controls the layout and appearance of your app. The server is where the logic of the app is implemented—for example, where calculations are performed and plots are generated.

An empty UI is created using the fluidPage() function. Adding text to a Shiny app is done by adding text inside fluidPage() as an argument. In fact, the entire UI is built by supplying the fluidPage() function with as many arguments as you want.

Instructions 100 XP Create a Shiny app that displays the text "Shiny is fun" by following these specific instructions:

Load the shiny package. Create the UI for the Shiny app using the fluidPage() function. Add the text "Shiny is fun" to the UI.

```
# Load the shiny package
library(shiny)

# Define UI for the application
ui <- fluidPage(
  # Add the text "Shiny is fun"
  "Shiny is fun"
)

# Define the server logic
server <- function(input, output) {}

# Run the application
shinyApp(ui = ui, server = server)
```

Shiny applications not supported in static R Markdown documents

Formatted text Shiny has many functions that can transform plain text into formatted text. Simply place text inside the h1() function to create a primary header (e.g. a title), h2() for a secondary header, strong() to make text bold, em() to make text italicized, or any of the other formatting functions.

You can also intermingle plain text and formatted text as much as you'd like—just remember to separate all the elements with commas!

Instructions 100 XP Place the following text inside the Shiny app:

The word "DataCamp" as a primary header. The words "Shiny use cases course" as a secondary header. The word "Shiny" in italics. The words "is fun" as bold text. After completing the instructions, your Shiny app should render the following text:

DataCamp Shiny use cases course Shiny is fun

```
# Load the shiny package
library(shiny)

# Define UI for the application
ui <- fluidPage(
  # "DataCamp" as a primary header
  h1("DataCamp"),
  # "Shiny use cases course" as a secondary header
  h2("Shiny use cases course"),
  # "Shiny" in italics
  em("Shiny"),
  # "is fun" as bold text
  strong("is fun")
)

# Define the server logic
server <- function(input, output) {}

# Run the application
shinyApp(ui = ui, server = server)
```

Shiny applications not supported in static R Markdown documents

Adding structure to your app Layouts in Shiny are used to give your app some structure by placing elements in certain desired positions.

A sidebar layout, created with the sidebarLayout() function, provides a basic two-column structure with a smaller sidebar on the left and a larger main panel on the right.

The sidebar layout function takes two arguments: sidebarPanel() and mainPanel(). Each of these panels can contain any arbitrary mix of text/HTML elements, in a similar fashion to how you can mix these elements inside a fluidPage().

Instructions 100 XP Your task is to add a sidebar layout to the existing app, such that the inputs will be on the left side and the outputs will be in the main panel. Specifically, you need to:

Define the UI for the Shiny application. Add a sidebar layout to the page. Add a sidebar panel to the layout, and place the inputs and text in it. Add a main panel to the layout, and place the plot and table in it.

```
# Load the shiny package
library(shiny)

# Define UI for the application
ui <- fluidPage(
  # Add a sidebar layout to the application
  sidebarLayout(
    # Add a sidebar panel around the text and inputs
```

```
    sidebarPanel(
      h4("Plot parameters"),
      textInput("title", "Plot title", "Car speed vs distance to stop"),
      numericInput("num", "Number of cars to show", 30, 1, nrow(cars)),
      sliderInput("size", "Point size", 1, 5, 2, 0.5)
    ),
    # Add a main panel around the plot and table
    mainPanel(
      plotOutput("plot"),
      tableOutput("table")
    )
  )
)

# Define the server logic
server <- function(input, output) {
  output$plot <- renderPlot({
    plot(cars[1:input$num, ], main = input$title, cex = input$size)
  })
  output$table <- renderTable({
    cars[1:input$num, ]
  })
}

# Run the application
shinyApp(ui = ui, server = server)
```

Shiny applications not supported in static R Markdown documents

Adding inputs Inputs are Shiny's way of allowing users to interact with an app. For example, textInput() is used to let the user enter text and numericInput() lets the user select a number. In the next chapter we will see many other types of inputs.

To add an input to your app, simply add the input function inside fluidPage(). Recall from the video that all input functions have the same first two arguments: inputId and label.

Instructions 100 XP Define the UI for the Shiny application. Create a numeric input with ID "age" and a descriptive label of "How old are you?". Create a text input with ID "name" and a label of "What is your name?".

```
library(shiny)

# Define UI for the application
ui <- fluidPage(
  # Create a numeric input with ID "age" and label of
  # "How old are you?"
  numericInput("age", "How old are you?", value = 20),

  # Create a text input with ID "name" and label of
  # "What is your name?"
  textInput("name", "What is your name?")
)

# Define the server logic
server <- function(input, output) {}
```

```
# Run the application
shinyApp(ui = ui, server = server)
```

Shiny applications not supported in static R Markdown documents

Adding placeholders for outputs Outputs are any object that should be displayed to the user and is generated in R, such as a plot or a table.

To add an output to a Shiny app, the first thing you need to do is add a placeholder for the output that tells Shiny where to place the output.

There are several output placeholder functions provided by Shiny, one for each type of output. For example, plotOutput() is for displaying plots, tableOutput() is for outputting tables, and textOutput() is for dynamic text.

Instructions 100 XP Create a text input with an ID of "name" in the sidebar panel. Add three output placeholders to the main panel: A text output with ID "greeting" (line 14). A plot output with ID "cars_plot" (line 16). A table output with ID "iris_table" (line 18).

```
library(shiny)

# Define UI for the application
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      # Create a text input with an ID of "name"
      textInput("name", "What is your name?", "Dean"),
      numericInput("num", "Number of flowers to show data for",
                   10, 1, nrow(iris))
    ),
    mainPanel(
      # Add a placeholder for a text output with ID "greeting"
      textOutput(outputId = "greeting"),
      # Add a placeholder for a plot with ID "cars_plot"
      plotOutput("cars_plot"),
      # Add a placeholder for a table with ID "iris_table"
      tableOutput("iris_table")
    )
  )
)

# Define the server logic
server <- function(input, output) {}

# Run the application
shinyApp(ui = ui, server = server)
```

Shiny applications not supported in static R Markdown documents

Constructing output objects There are three rules to build an output in Shiny:

Build the object with the appropriate render*() function.

Save the result of the render function into the output list, which is a parameter of the server function. Specifically, save it into output$ in order to replace the output placeholder in the UI that has ID outputId.

If the output relies on any user-modified input values, you can access any of the inputs using the input parameter of the server function. Specifically, input$ will always return the current value of the input field that has ID inputId.

Instructions 100 XP You are given a Shiny app with a fully functional UI portion. Your task is to construct all the outputs. Specifically:

Create a plot of the cars dataset in the plot output placeholder with ID "cars_plot" (line 23). In the "greeting" text output, render a text greeting in the form of "Hello NAME", where NAME is the value of the name input (line 28). In the "iris_table" output, show a table of the first n rows of the iris dataset, where n is the value of the numeric input (line 33).

```
# Load the shiny package
library(shiny)

# Define UI for the application
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      textInput("name", "What is your name?", "Dean"),
      numericInput("num", "Number of rows to show in the table",
                   10, 1, nrow(iris))
    ),
    mainPanel(
      textOutput("greeting"),
      plotOutput("cars_plot"),
      tableOutput("iris_table")
    )
  )
)

# Define the server logic
server <- function(input, output) {
  # Create a plot of the "cars" dataset
  output$cars_plot <- renderPlot({
    plot(cars)
  })

  # Render a text greeting as "Hello <name>"
  output$greeting <- renderText({
    paste("Hello", input$name)
  })

  # Show a table of the first n rows of the "iris" data
  output$iris_table <- renderTable({
    data <- iris[1:input$num, ]
    data
  })
}

# Run the application
shinyApp(ui = ui, server = server)
```

Shiny applications not supported in static R Markdown documents

Reactive contexts Reactive values are special constructs in Shiny; they are not seen anywhere else in R programming. As such, they cannot be used in just any R code, reactive values can only be accessed within a reactive context.

This is the reason why any variable that depends on a reactive value must be created using the reactive() function, otherwise you will get an error. The shiny server itself is not a reactive context, but the reactive() function, the observe() function, and all render*() functions are.

Instructions 100 XP You are provided with a Shiny app containing two numeric inputs, num1 and num2, and a text output. Your task is to:

In a reactive variable called my_sum, calculate the sum of the two numeric inputs (line 10). In a reactive variable called my_average, calculate the average of the two inputs (line 14). In the text output, display the calculated average using the reactive variables (line 23).

```
ui <- fluidPage(
  numericInput("num1", "Number 1", 5),
  numericInput("num2", "Number 2", 10),
  textOutput("result")
)

server <- function(input, output) {
  # Calculate the sum of the inputs
  my_sum <- reactive({
    input$num1 + input$num2
  })

  # Calculate the average of the inputs
  my_average <- reactive({
    my_sum() / 2
  })

  output$result <- renderText({
    paste(
      # Print the calculated sum
      "The sum is", my_sum(),
      # Print the calculated average
      "and the average is", my_average()
    )
  })
}

shinyApp(ui, server)
```

Shiny applications not supported in static R Markdown documents

Perfect Plot

More exploration of the Gapminder data To use the gapminder data in a Shiny app, you will often have to filter the dataset in order to retain only a subset of the rows. You can use the subset() function for that.

Instructions 100 XP You are given a Shiny app that contains an empty text output. Your task is to:

Load the gapminder package. Determine the population of France in year 1972 using the subset() function, and display that number in the text output (line 14).

```r
# Load the gapminder package
library(gapminder)

# Define UI for the application
ui <- fluidPage(
  "The population of France in 1972 was",
  textOutput("answer")
)

# Define the server function
server <- function(input, output) {
  output$answer <- renderText({
    # Determine the population of France in year 1972
    subset(gapminder, country == "France" & year == 1972)$pop
  })
}

# Run the application
shinyApp(ui = ui, server = server)
```

Shiny applications not supported in static R Markdown documents

Add a plot title: text input In Shiny, as soon as the user changes the value of any input, Shiny makes the current value of that input immediately available to you in the server through the input argument of the server function. You can retrieve the value of any input using input$.

In order to assign a default initial value to a text input, the value argument is used.

Instructions 100 XP The given Shiny app plots the GDP per capita vs life expectancy of countries in the gapminder dataset. Your task is to add a text input that lets users change the title of the plot. Specifically:

Add a text input to the UI with ID “title”, a label of “Title”, and a default value of “GDP vs life exp”. In the server code, make the title of the plot always reflect the current value of the title input by placing the title inside the ggtitle() function (line 24).

```r
# Load the ggplot2 package for plotting
library(ggplot2)

# Define UI for the application
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      # Add a title text input
      textInput("title", "Title", "GDP vs life exp")
    ),
    mainPanel(
      plotOutput("plot")
    )
  )
)

# Define the server logic
server <- function(input, output) {
  output$plot <- renderPlot({
    ggplot(gapminder, aes(gdpPercap, lifeExp)) +
```

```
      geom_point() +
      scale_x_log10() +
      # Use the input value as the plot's title
      ggtitle(input$title)
  })
}

# Run the application
shinyApp(ui = ui, server = server)
```

Shiny applications not supported in static R Markdown documents