Barcelona R

Introduction to Shiny

Nicolas Attalides

Who am I?

- **Name:** Nicolas Attalides
- **Coding in R since:** 2005 (yes that's before RStudio!)
- **Profession:** Data Scientist consultant and trainer (5+ yrs.)
- **Education:** PhD in Statistical Science from UCL (2015)
- **R Status:** A never-ending evolving R dinosaur
- **Hobbies:** Tennis and coding (not at the same time)

Workshop Setup:

- Wi-Fi

Network Name: N/A

Password: N/A

- Resources

R (version 3.6.3)

RStudio (version 1.3.959)

Shiny (version 1.5.0)

What is Shiny?

Shiny is an R package that allows you to design and build interactive web applications using R.

You do not need to know how to code in HTML, CSS, or JavaScript.

Shiny is easy to write and one of the best ways to let users interact with data.

Topics

- Workshop aim:

Learn how to develop a simple Shiny app.

- Topics:
  - User Interface and Server scripts
  - Inputs and Outputs
  - Widgets and Reactivity
  - How to share your app

User Interface and Server scripts

ui.R (User Interface script)

- Designs and structures the layout of the application.
- Defines what the user sees and interacts with.

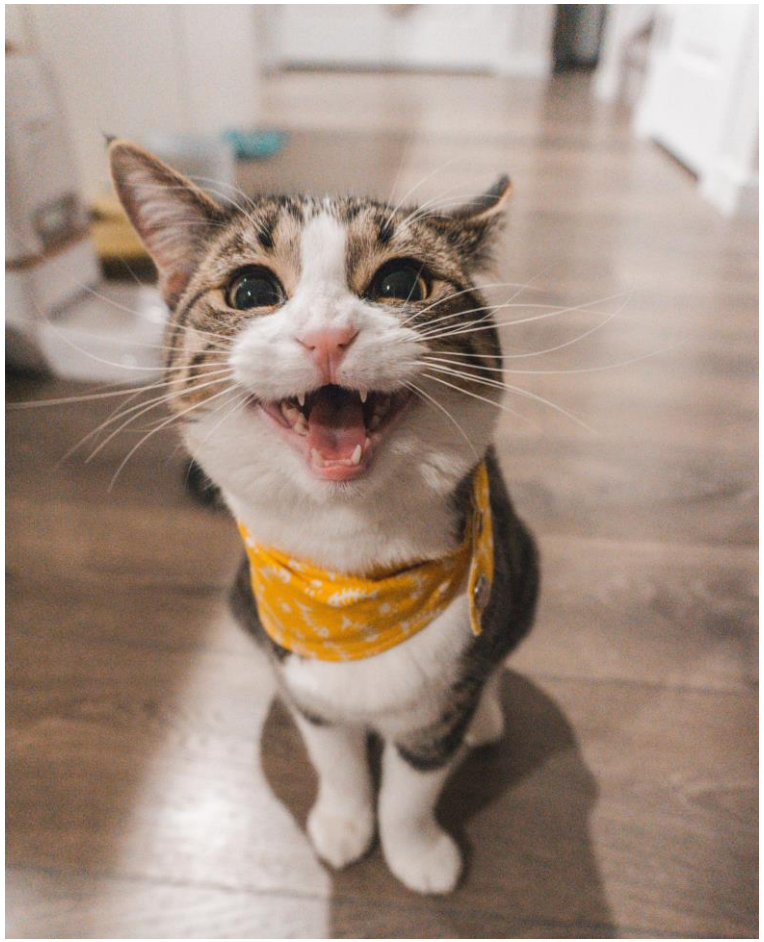
server.R (Server script)

- Defines the server-side logic of the application.
- Contains the R code and other instructions to execute.

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

You can press the Run App button to start the app!

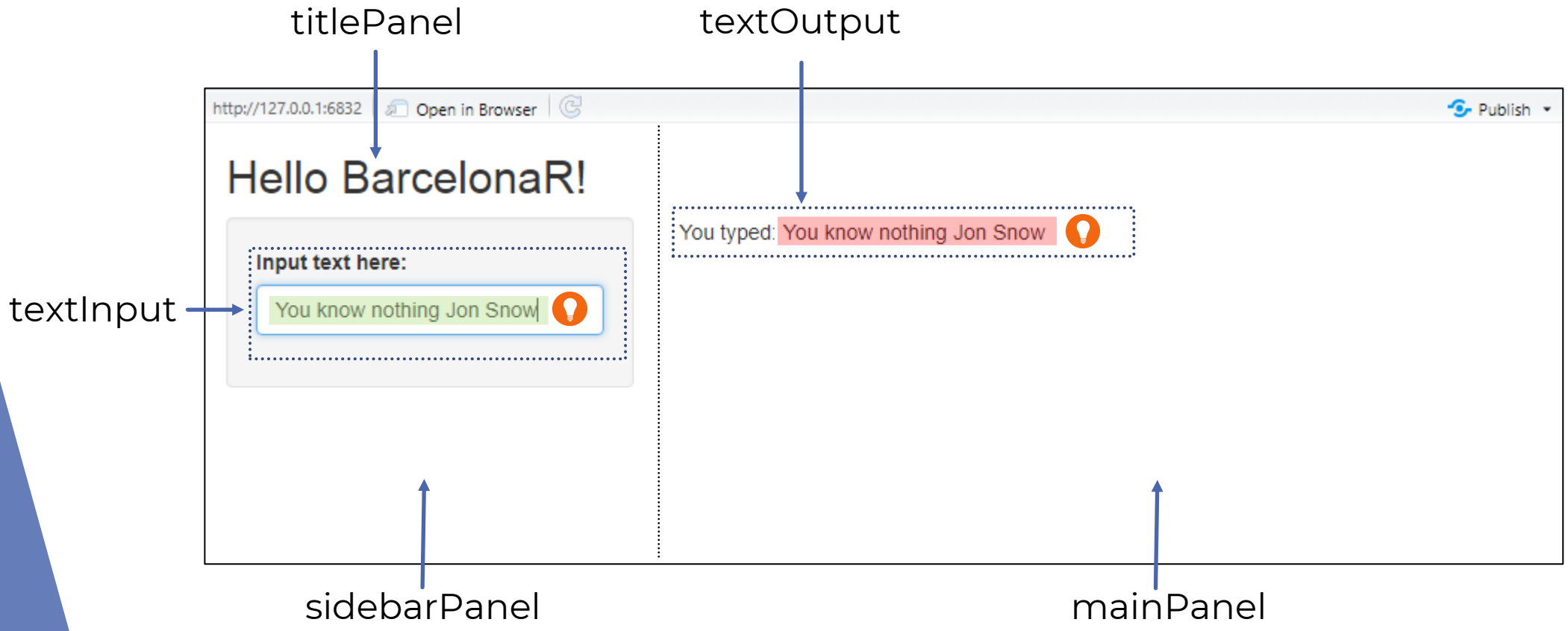
Live Coding Example 1



Create a Shiny app with a text input widget that displays what the user writes.

1. In the UI script create a textInput widget to allow the user to input text.
2. In the server script use the input to display the text that the user writes.

Live Coding Example 1



input

output

Live Coding Example 1 – ui.R

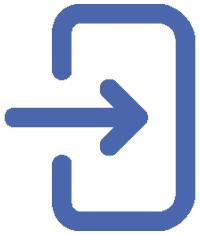
```
# Define UI for application
ui <- fluidPage(

  # Application title
  titlePanel("Hello BarcelonaR!"),

  # Sidebar with an input
  sidebarLayout(
    sidebarPanel(
      textInput("text_input", "Input text here:")
    ),

    # Main with output
    mainPanel(
      textOutput("text_output")
    )
  )
)
```

Live Coding Example 1 – server.R

```
library(shiny)

# Define server logic and R code
server <- function(input, output) {

  output$text_output <- renderText({
    # Display text input
    paste("You typed:", input$text_input)
  })
}
```

Inputs and Outputs

Inputs



ui.R →

```
textInput("text_input", "Input text here:")
```

server.R →

```
paste("You typed:", input$text_input)
```

Outputs

server.R →

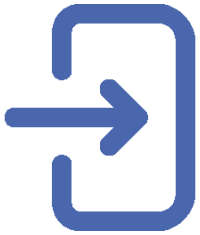
```
output$text_output <- renderText({ })
```

ui.R →

```
textOutput("text_output")
```

Inputs and Outputs

Inputs



`textInput()`

Text input

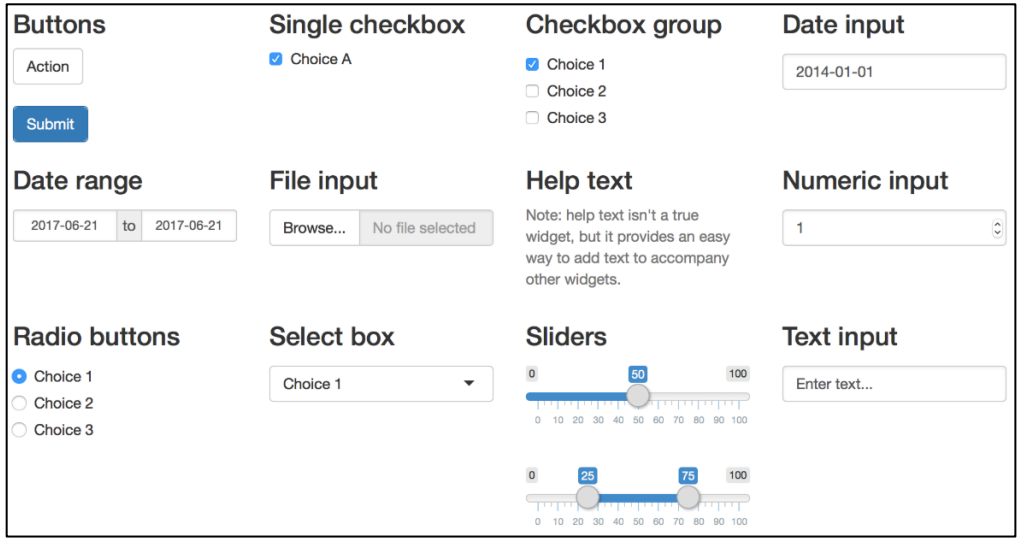
Enter text...

Outputs

`textOutput()`

Widgets

function	widget
actionButton	Action Button
submitButton	A submit button
checkboxInput	A single check box
dateInput	A calendar to aid date selection
numericInput	A field to enter numbers
radioButtons	A set of radio buttons
selectInput	A box with choices to select from
sliderInput	A slider bar
textInput	A field to enter text



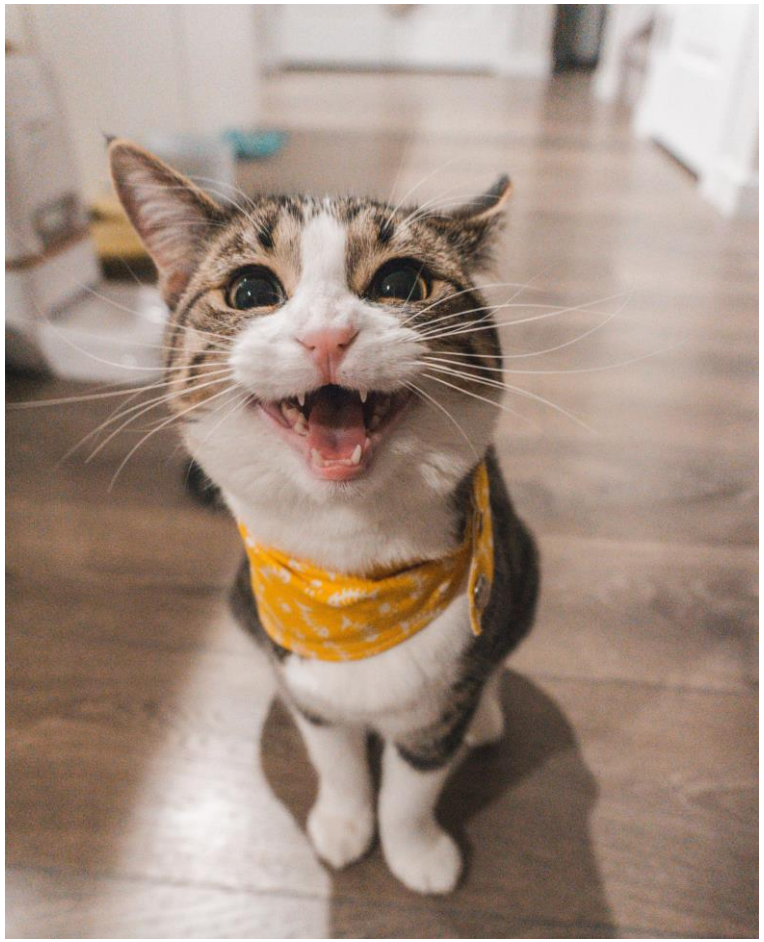
https://shiny.rstudio.com/tutorial/written-tutorial/lesson3/

Functions

ui.R Output function	server.R render function	Creates
dataTableOutput	renderDataTable	DataTable
imageOutput	renderImage	image
plotOutput	renderPlot	plot
tableOutput	renderTable	table
textOutput	renderText	text
uiOutput	renderUI	raw HTML

https://shiny.rstudio.com/tutorial/written-tutorial/lesson4/

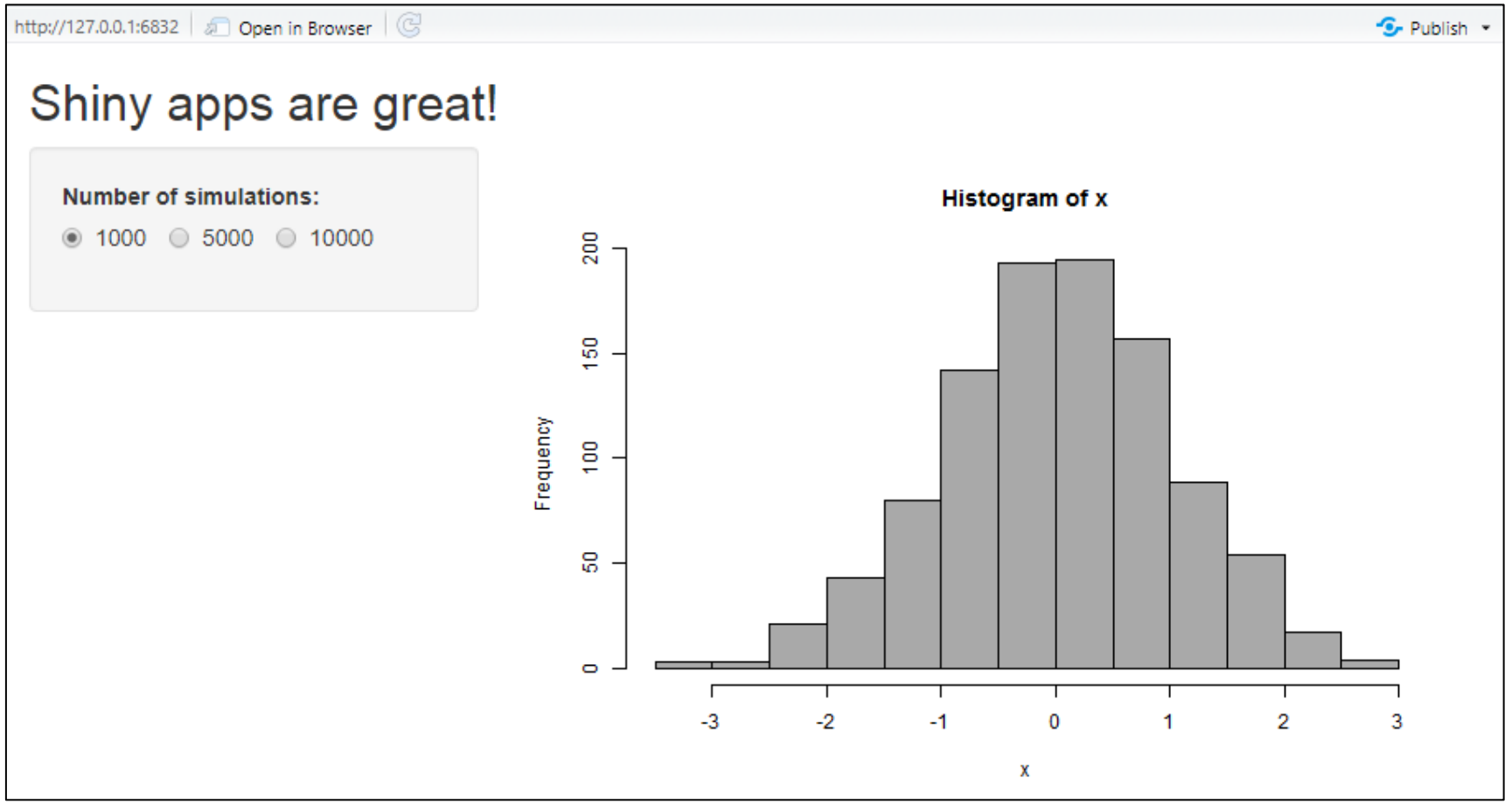
Live Coding Example 2



Create a Shiny app with a radioButton widget and a histogram plot of random standard normal distribution values.

1. In the UI script create a radioButton widget to allow the user to choose 1000, 5000 or 10000 simulations to generate.
2. In the server script use the input to generate the number of simulations and then render the output plot of the histogram.

Live Coding Example 2



Live Coding Example 2 – ui.R

```
ui <- fluidPage(

  titlePanel("Shiny apps are great!"),

  sidebarLayout(
    sidebarPanel(
      radioButtons(inputId = "number",
                   label = "Number of simulations:",
                   choices = c(1000, 5000, 10000),
                   inline = TRUE) # put radio buttons in horizontal line
    ),

    mainPanel(
      plotOutput("distPlot")
    )
  )
)
```

Live Coding Example 2 – server.R

```
library(shiny)

server <- function(input, output) {

  output$distPlot <- renderPlot({
    # generate simulations based on input$number from ui.R
    x <- rnorm(input$number)

    hist(x, col = 'darkgray')
  })
}
```

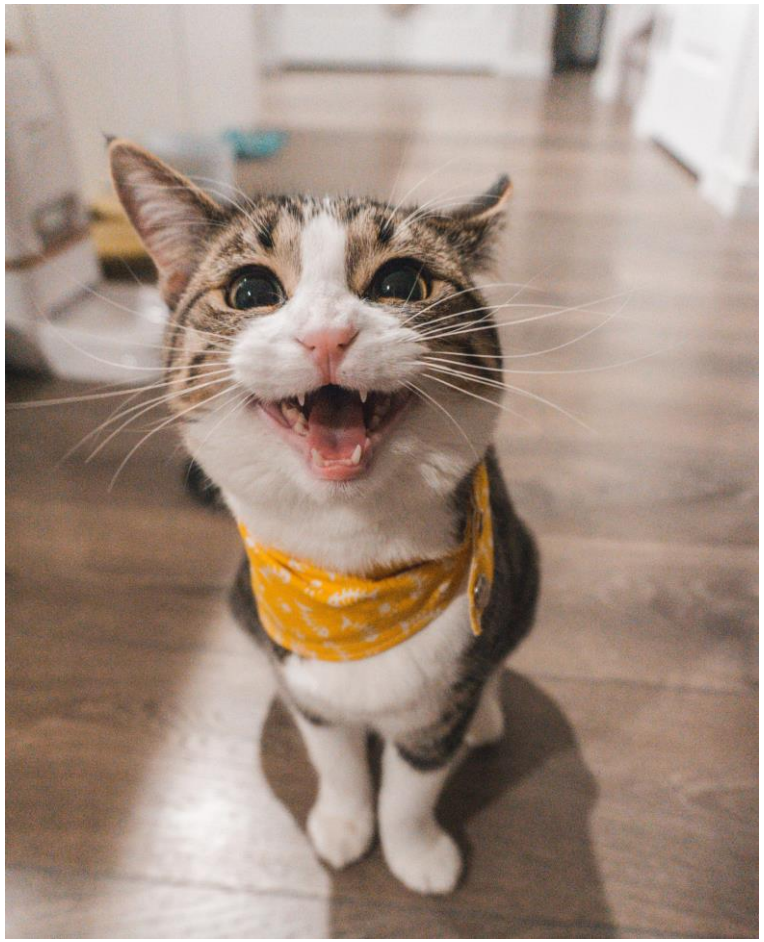
Reactivity

This is what makes a Shiny app responsive to user interactions.

- It happens automatically when the values of inputs are changed.
- The updated input values are passed on to the server.
- The server executes and runs any R code relating to the inputs.
- The updated outputs are rendered and returned to the app.

The list object `input$` contains the values for each input.

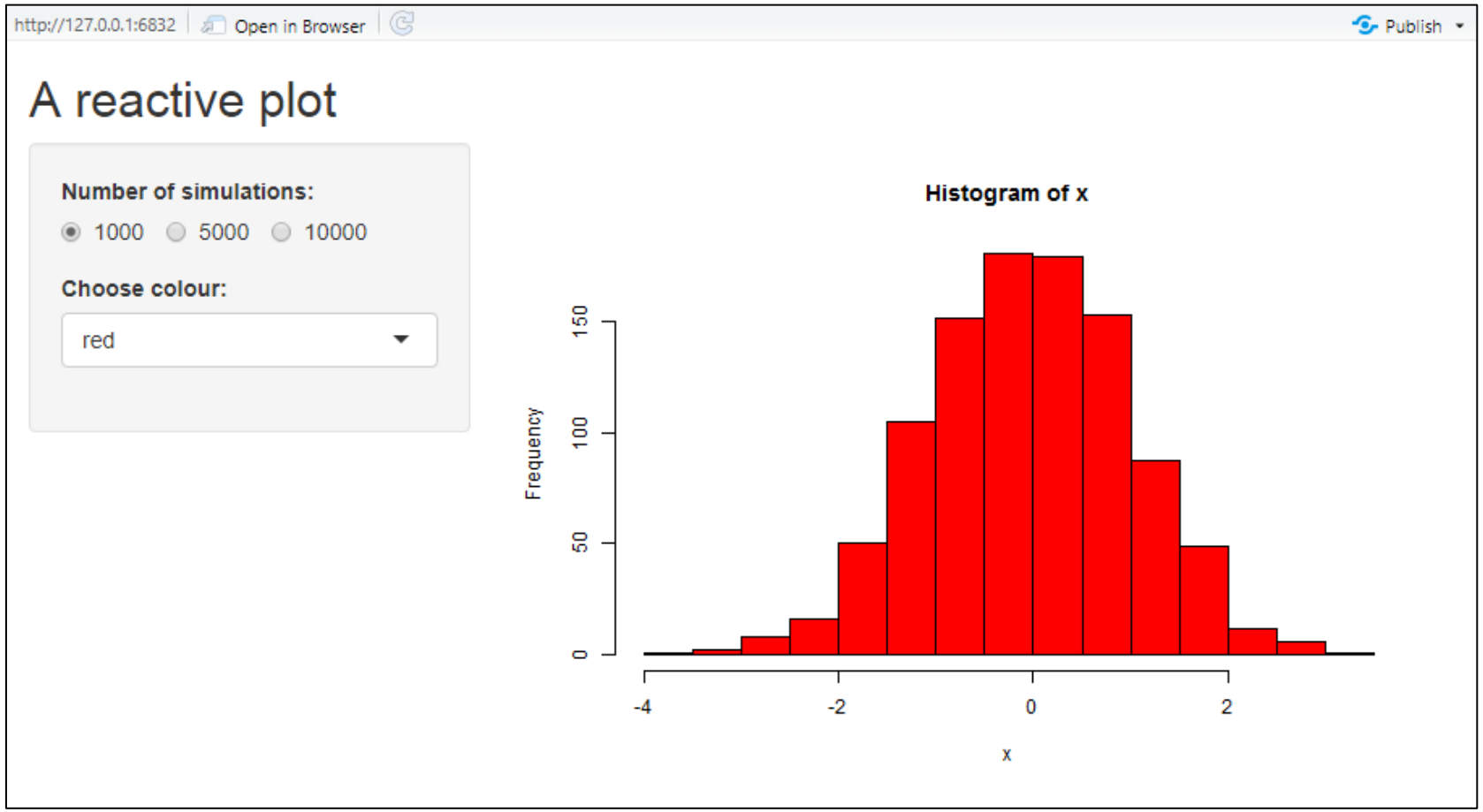
Live Coding Example 3



Add a drop down list to the Shiny app which allows the user to select the colour of the histogram plot.

1. In the UI script create a selectInput widget to allow the user to select “red”, “blue” or “green” for the histogram plot.
2. Edit the server script to use the new input when rendering the output plot for the histogram.

Live Coding Example 3



Changing the colour re-evaluates the simulation!

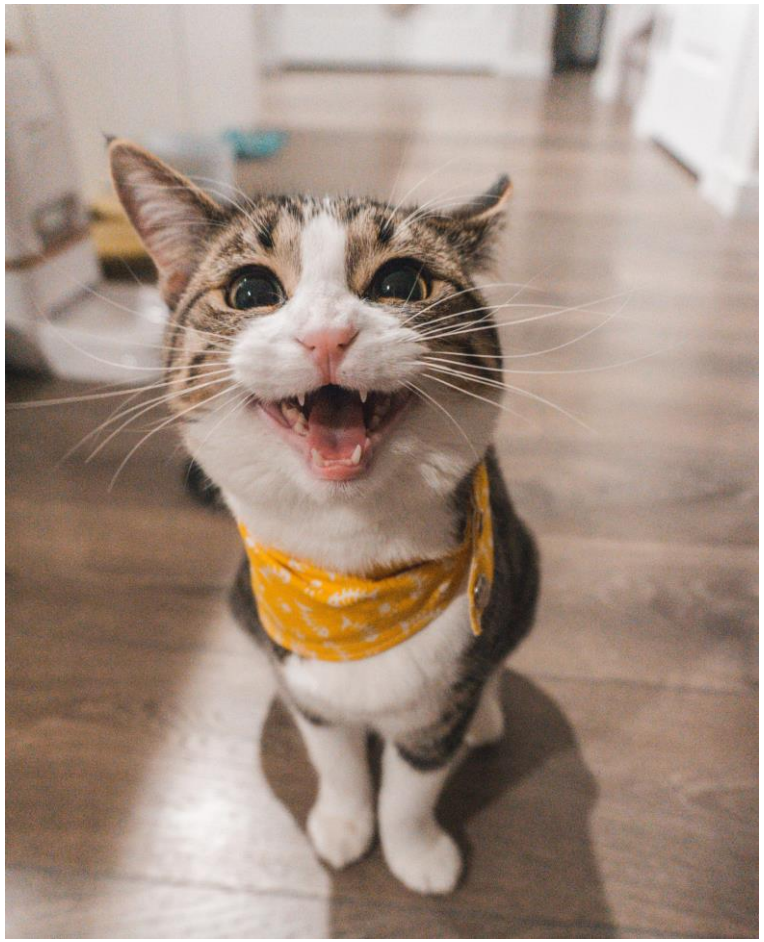
Live Coding Example 3 – ui.R

```
ui <- fluidPage(

  titlePanel("A reactive plot"),

  sidebarLayout(
    sidebarPanel(
      radioButtons(inputId = "number",
                   label = "Number of simulations:",
                   choices = c(1000, 5000, 10000),
                   inline = TRUE),
      selectInput(inputId = "colour",
                  label = "Choose colour:",
                  choices = c("red", "blue", "green"))
    ),
    mainPanel(
      plotOutput("distPlot")
    )
  )
)
```

Live Coding Example 3 – server.R

```
library(shiny)

server <- function(input, output) {

  output$distPlot <- renderPlot({
    # generate simulations based on input$number from ui.R
    x <- rnorm(input$number)

    hist(x, col = input$colour)
  })
}
```

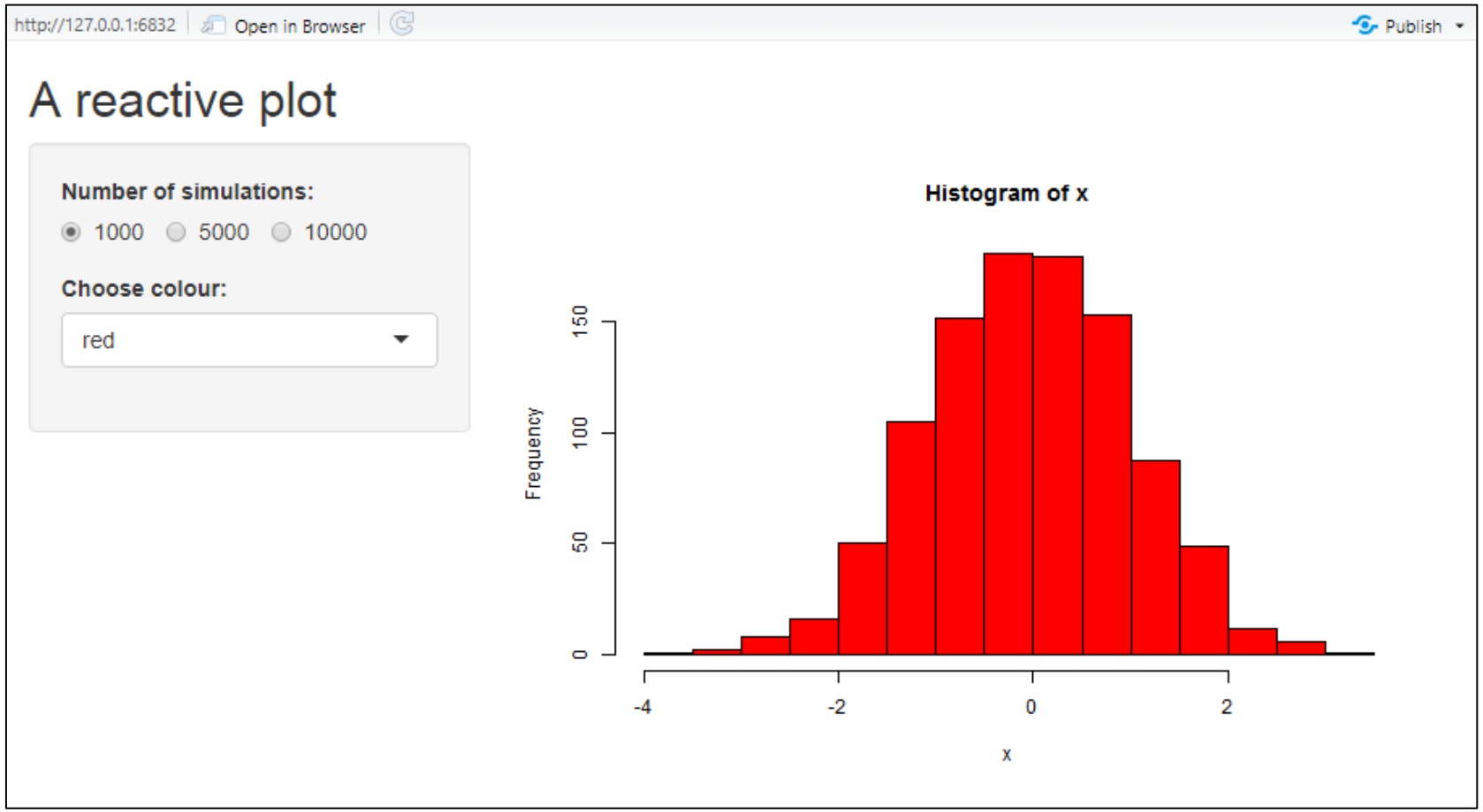
Live Coding Example 4



Add a reactive expression to stop the simulation from re-evaluating each time the colour input is updated.

1. In the server script use the `reactive` function with the radioButton input to create the dataset that is passed to the output plot.

Live Coding Example 4



Live Coding Example 4 – ui.R

```
ui <- fluidPage(

  titlePanel("A reactive plot"),

  sidebarLayout(
    sidebarPanel(
      radioButtons(inputId = "number",
                   label = "Number of simulations:",
                   choices = c(1000, 5000, 10000),
                   inline = TRUE),
      selectInput(inputId = "colour",
                  label = "Choose colour:",
                  choices = c("red", "blue", "green"))
    ),
    mainPanel(
      plotOutput("distPlot")
    )
  )
)
```

No need to change the UI!

Live Coding Example 4 – server.R

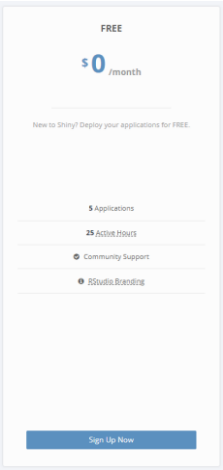
```
library(shiny)

server <- function(input, output) {

  sim_data <- reactive({
    rnorm(input$number)
  })

  output$distPlot <- renderPlot({
    x <- sim_data()

    hist(x, col = input$colour)
  })
}
```

To use a reactive object you need to include the parentheses!

How to share your app

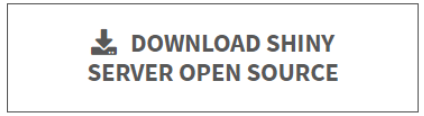
- Shinyapps.io https://www.shinyapps.io/

An easy way to share you application that is secure and scalable utilising a server that is maintained by RStudio. There is a free tier available!

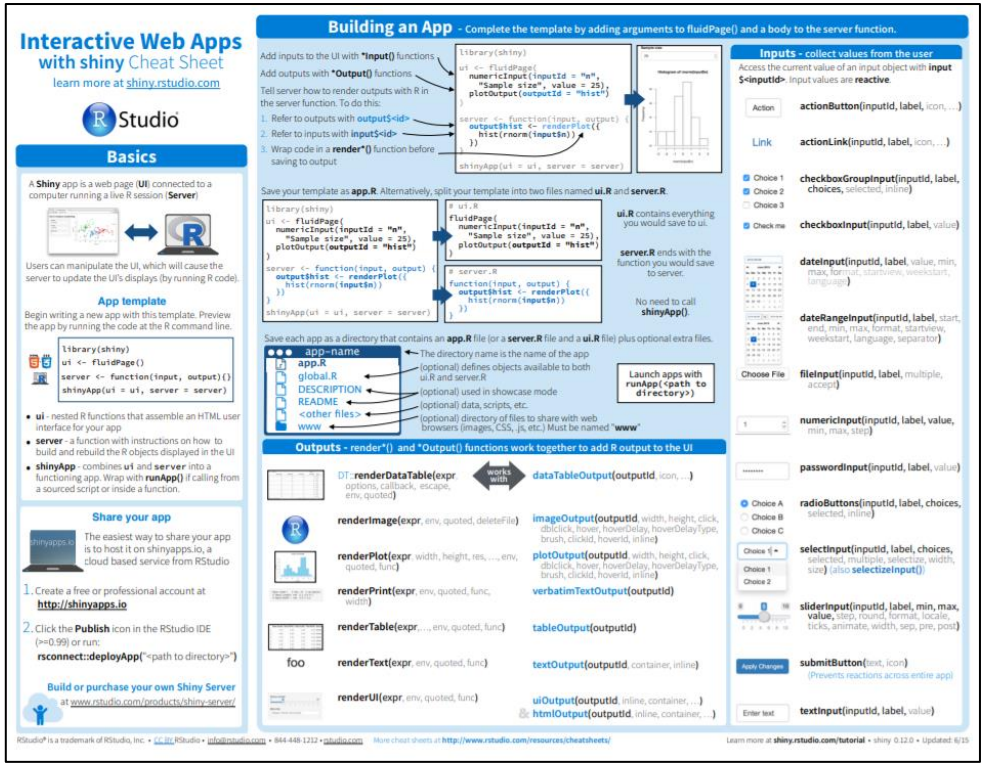


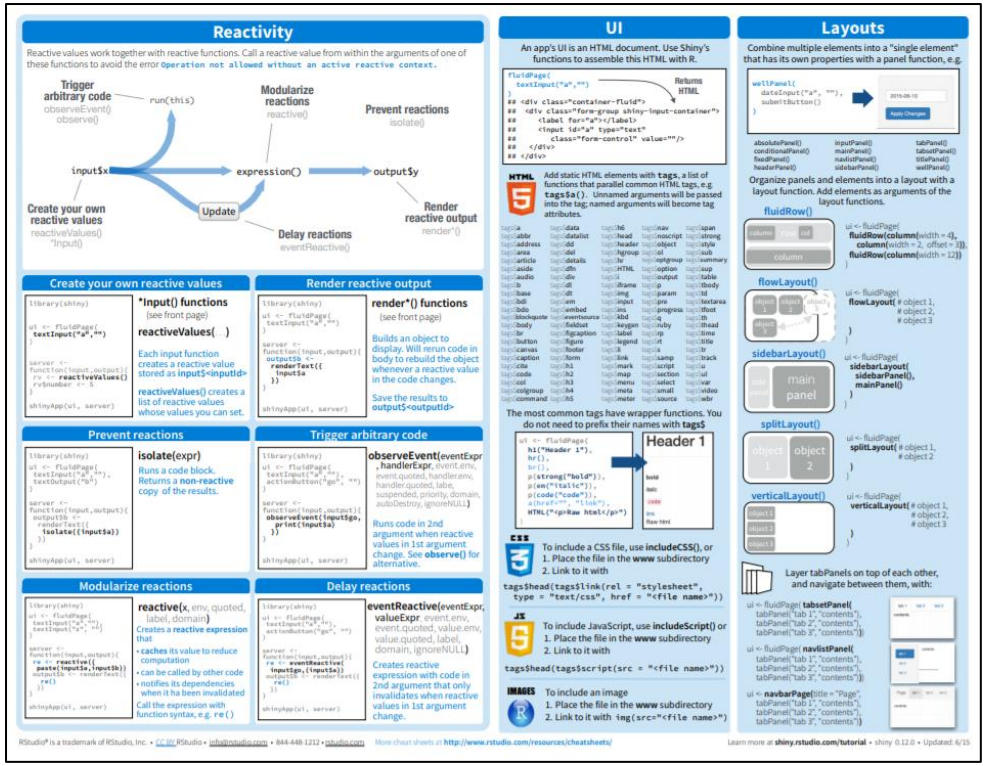
- Shiny Server Open Source https://www.rstudio.com/products/shiny/shiny-server/

You can have your own server to host your applications. This also allows you to customise each app to have its own URL.



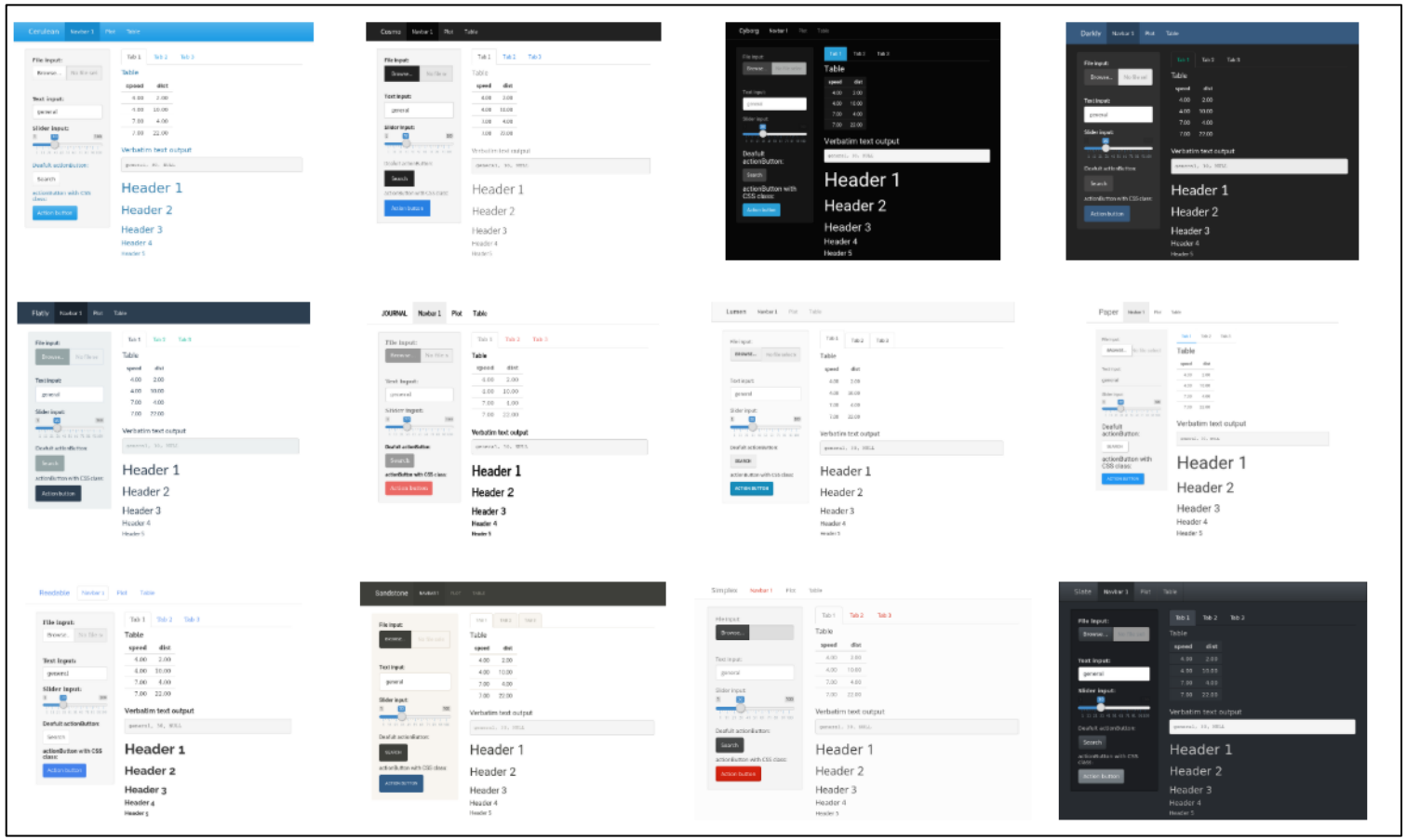
Other resources – Shiny cheat sheet





https://shiny.rstudio.com/images/shiny-cheatsheet.pdf

Other resources – Shiny Themes



https://rstudio.github.io/shinythemes/

Thank you to our sponsors and partners!