RStudio

250 Northern Ave, Boston, MA 02210
Phone: 844-448-1212

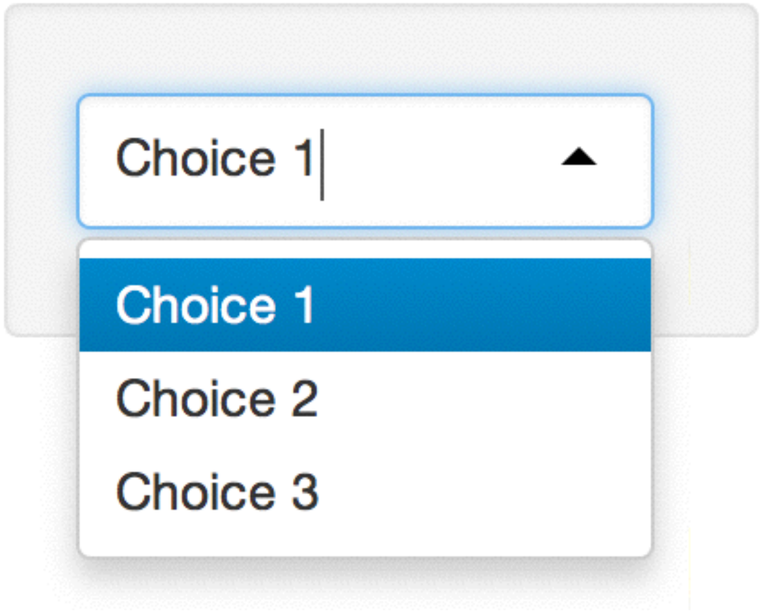
Email: info@rstudio.com
Web: http://www.rstudio.com

All Training materials are provided "as is" and without warranty and RStudio disclaims any and all express and implied warranties including without limitation the implied warranties of title, fitness for a particular purpose, merchantability and noninfringement.

250 Northern Ave, Boston, MA 02210
Phone: 844-448-1212
Email: info@rstudio.com
Web: http://www.rstudio.com

Reproducible expertise with Shiny

Empower your audience



Joe Cheng
Software Engineer
September 2015
Email: joe@rstudio.com

Reproducible data science

1 **Code**, results, and reasoning — **R Markdown** ✔

2 **Computing Environment** (R version, packages) — **Packrat** (projects) ✔

3 **Expertise** to extend, reapply, and reuse code — **Shiny** ✔

What is Shiny?

Motivation

R is a powerful platform for data analysis

- State of the art statistical power
- Excellent for visualization
- Large and enthusiastic community

Motivation

But…

- A personal experience, not a shared one
- Output is usually in static formats
- Modern visualization is interactive, browser-based

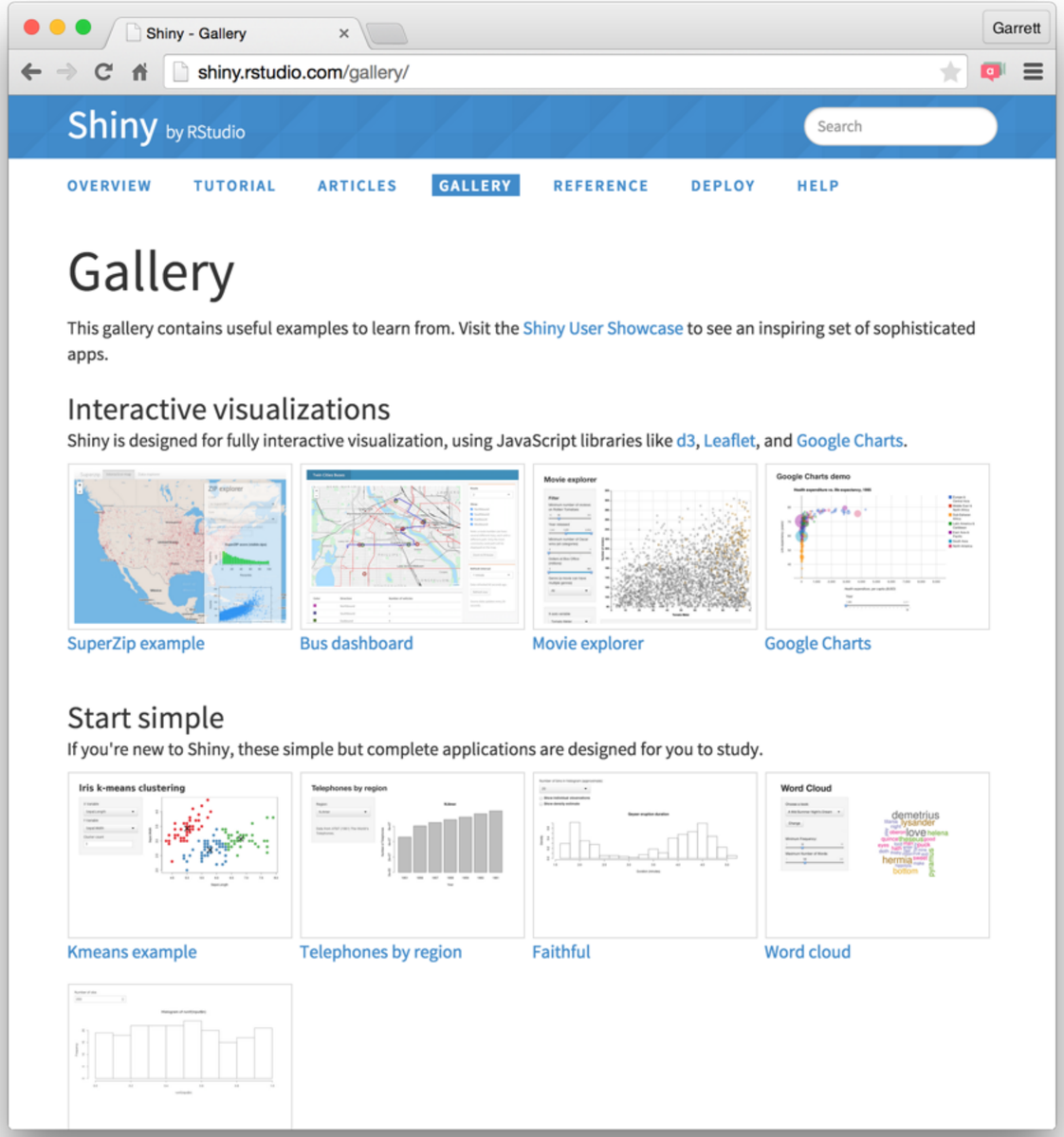
What is Shiny?

- Interactive web applications around your R analyses
- Zero HTML/CSS/JavaScript knowledge is required…
- …but fully customizable and extensible with HTML/CSS/JavaScript

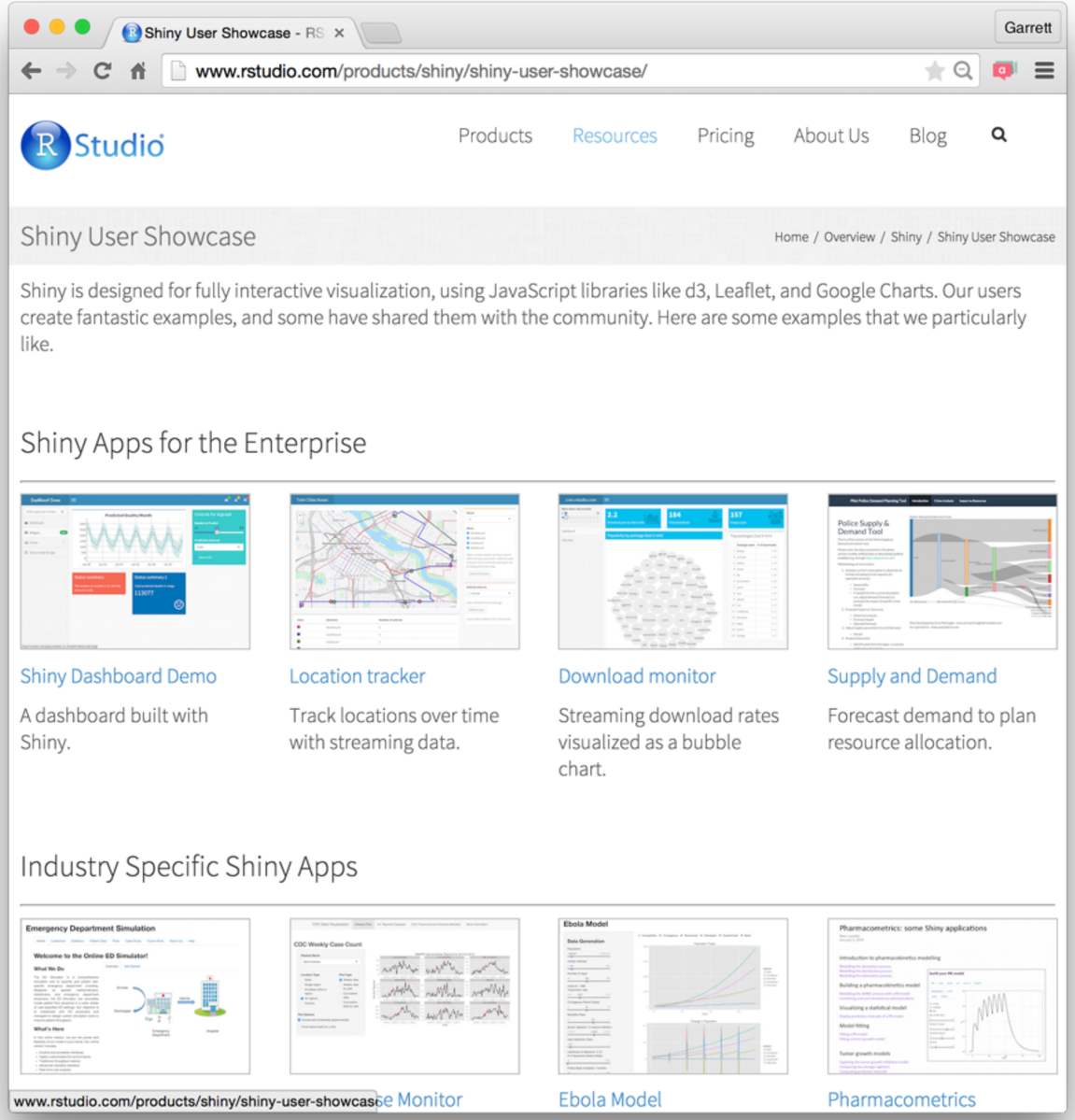
What is Shiny?

- Modern web UI with attractive defaults, but also infinitely customizable
- Designed to integrate with existing JavaScript libraries
- Uses a reactive programming model which allows dramatically simpler code than traditional UI or web programming

http://shiny.rstudio.com/gallery/kmeans-example.html



shiny.rstudio.com/gallery



http://www.rstudio.com/products/shiny/shiny-user-showcase/

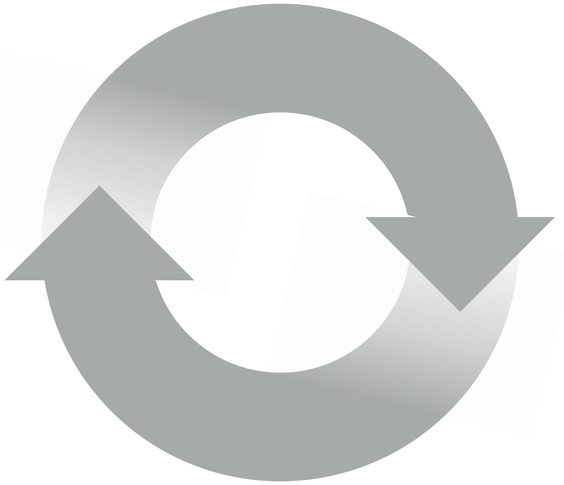
Shiny

Computation using R

+

Web-based user interface

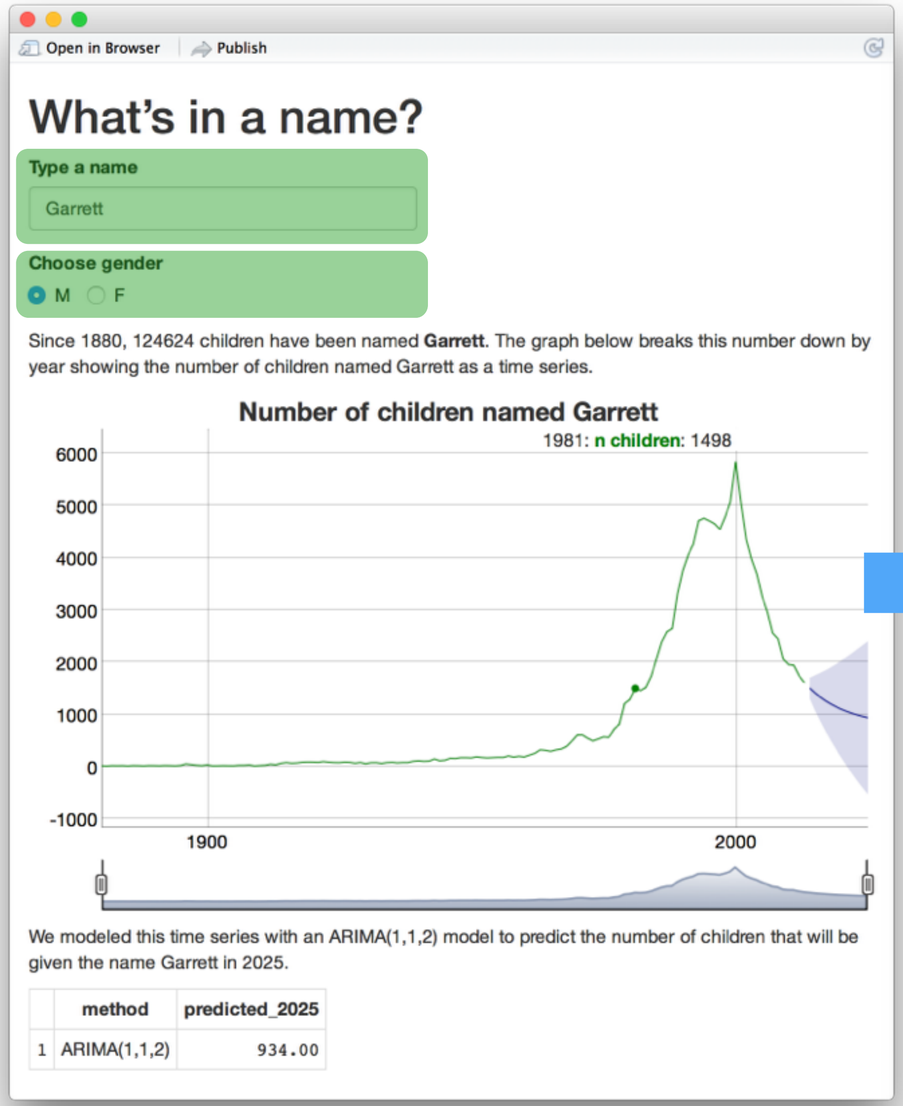
+

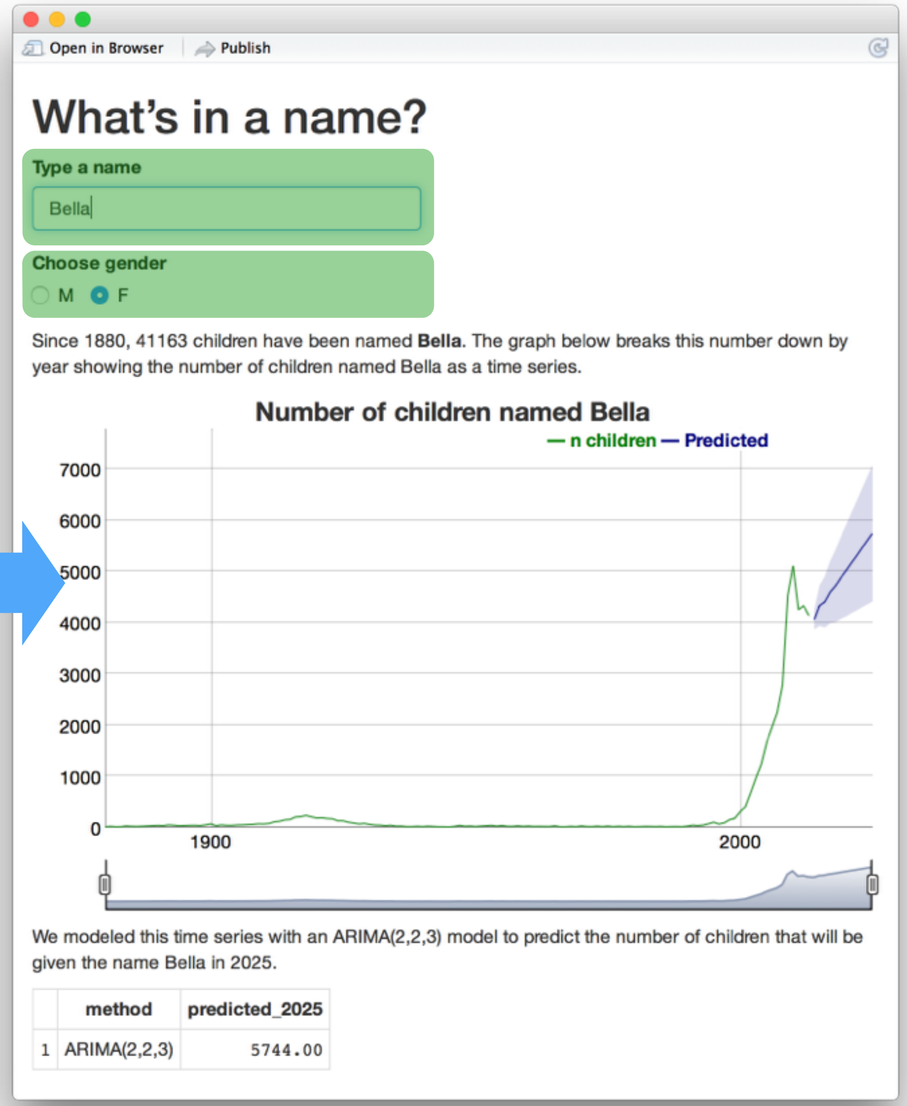


Reactive programming

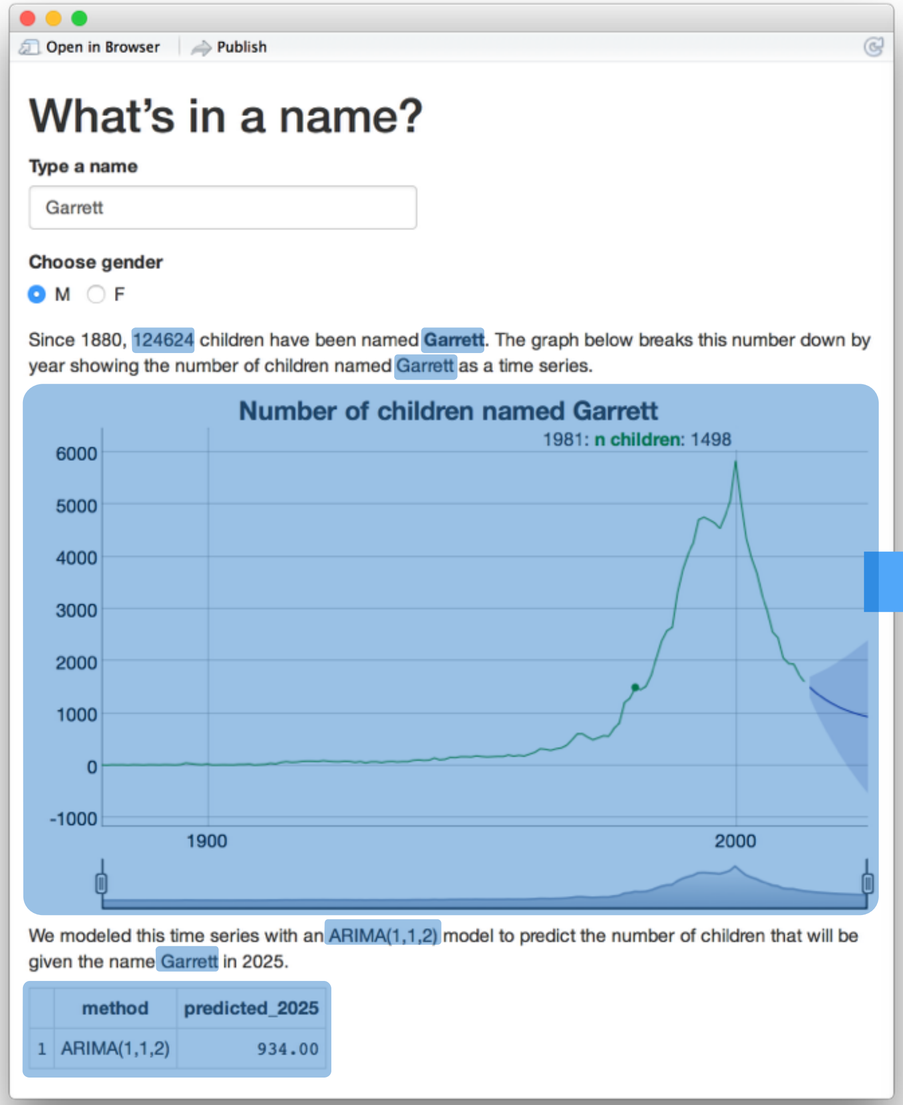
https://gallery.shinyapps.io/names-app/name-app.Rmd

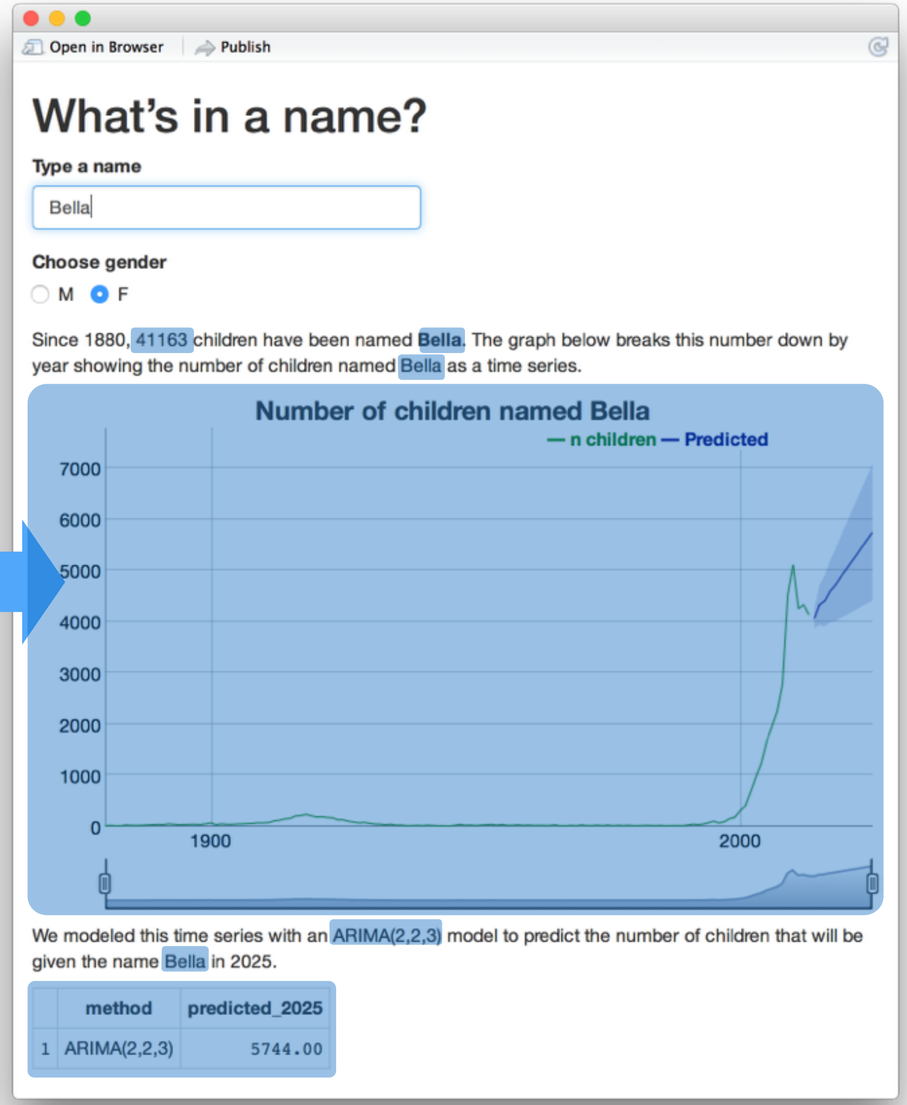
Goal: make display output react to user *input*





Goal: make display *output* react to user input





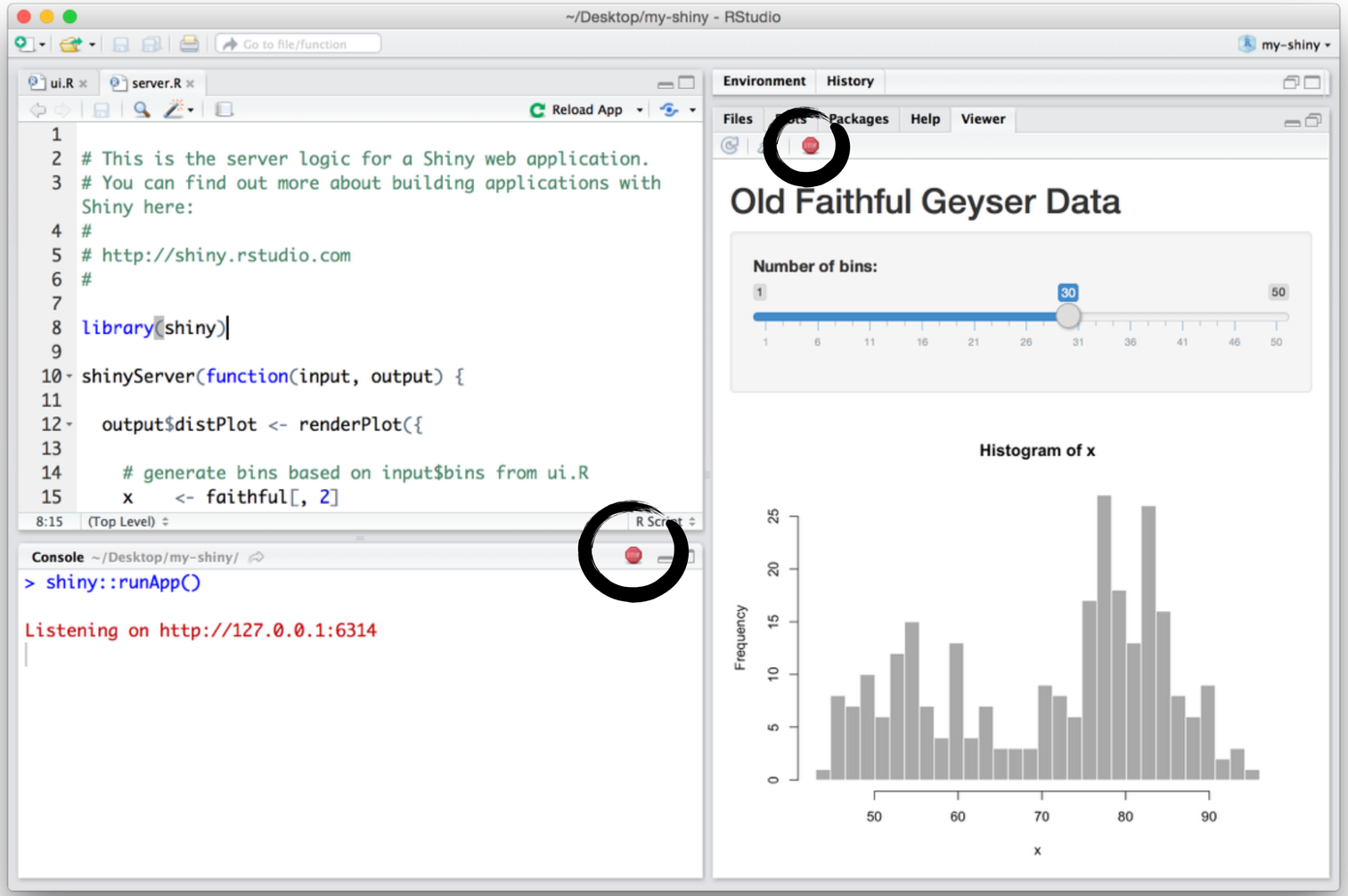
App template

App template

The shortest viable shiny app

```
library(shiny)
ui <- fluidPage()

server <- function(input, output) {}

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

Close an app



shinyApp

shinyApp() assembles the server and ui components into a complete app.

```
shinyApp(ui = my_ui, server = my_serv)
```

A ui object
that builds a web page
for your app

A server function
that builds the R
components of the app

Warm up

`exercise(20)` opens a copy of the app template. Launch the app by copying and pasting the code into the command line.

ui

1

ui contains a function that writes an HTML page for your app.

ui starts with the following code (or its equivalent).

```
fluidPage()
```

2

Add elements to your app as arguments to `fluidPage()`

```
fluidPage(
  # input controls,
  # output controls,
  # layout objects,
  # custom HTML/CSS/JavaScript
)
```

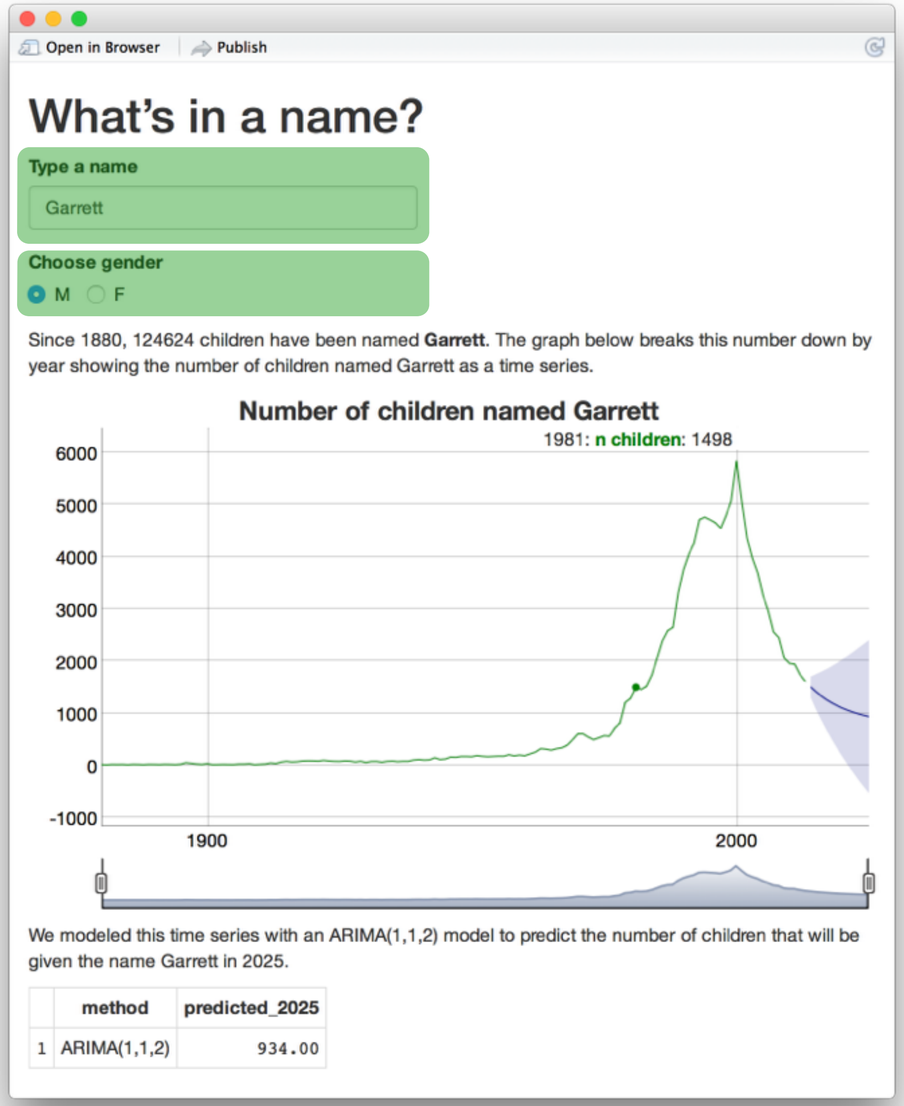
2

Add elements to your app as arguments to `fluidPage()`

```
fluidPage(
  # input controls,
  # output controls,
  # layout objects,
  # custom HTML/CSS/JavaScript
)
```

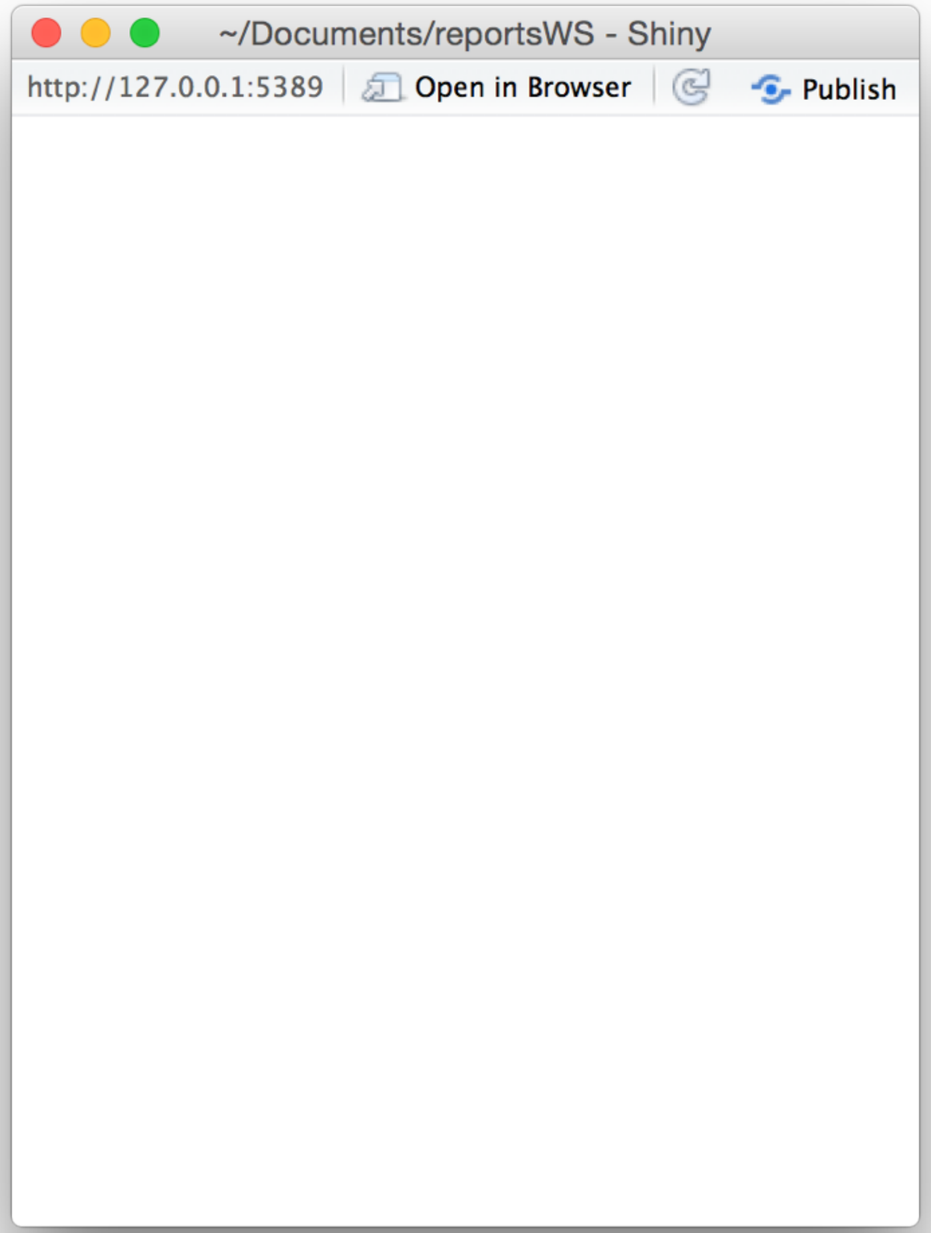
Inputs

Inputs - let users set a value by typing, clicking, etc.



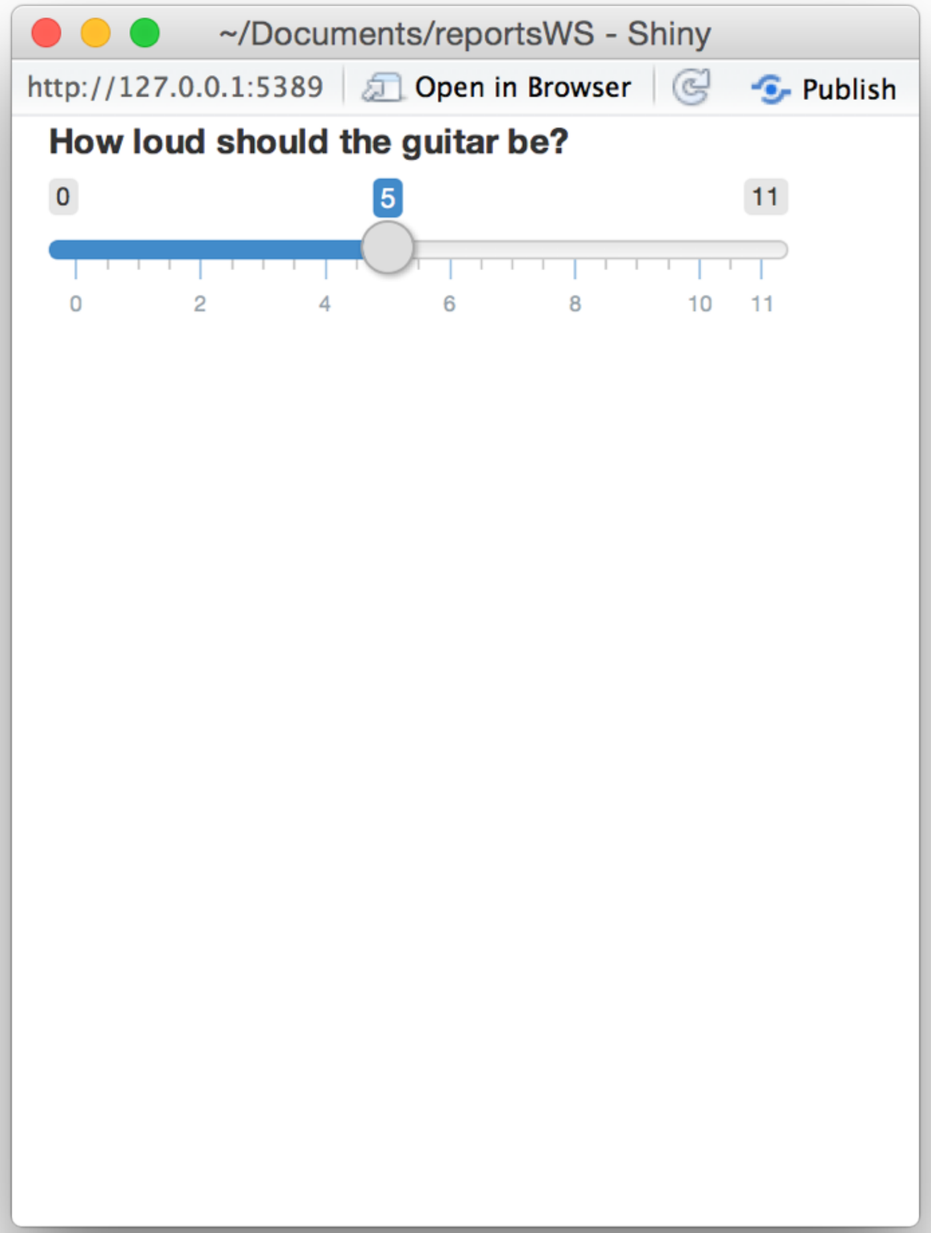
Create an input with an input function.

```
library(shiny)
ui <- fluidPage(




)

server <- function(input, output) {}

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



Create an input with an input function.

```
library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "sl",
    "How loud should the guitar be?",
    min = 0, max = 11, value = 5)
)

server <- function(input, output) {}

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



Buttons

Action

Submit

actionButton
submitButton

Single checkbox

Choice A

checkboxInput

Checkbox group

Choice 1
Choice 2
Choice 3

checkboxGroupInput

Date input

2014-01-01

dateInput

Date range

2014-01-24 to 2014-01-24

dateRangeInput

File input

Choose File No file chosen

fileInput

Numeric input

1

numericInput

Password Input

passwordInput

Radio buttons

Choice 1
Choice 2
Choice 3

radioButtons

Select box

Choice 1

selectInput

Sliders

sliderInput

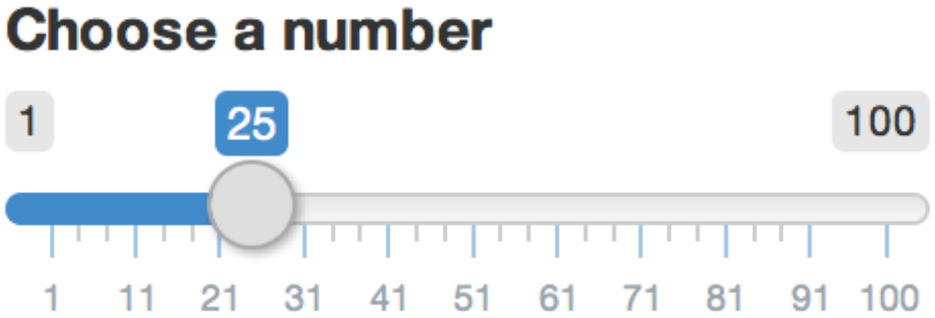
Text input

Enter text...

textInput

function	input object
`actionButton`	Action button
`actionLink`	Action link
`checkboxGroupInput`	Group of checkboxes
`checkboxInput`	Single checkbox
`dateInput`	Calendar to aid date selection
`dateRangeInput`	Pair of calendars for selecting date range
`fileInput`	File upload control wizard
`numericInput`	Field to enter numbers
`passwordInput`	Field to enter hidden text
`radioButtons`	Set of radio buttons
`selectInput`	Box with choices to select from
`sliderInput`	Slider bar
`textInput`	Field to enter text

Syntax




```
sliderInput(inputId = "num", label = "Choose a number", …)
```

- input function
- input name (for internal use)
- Notice: Id not ID
- label to display
- input specific arguments

Your Turn

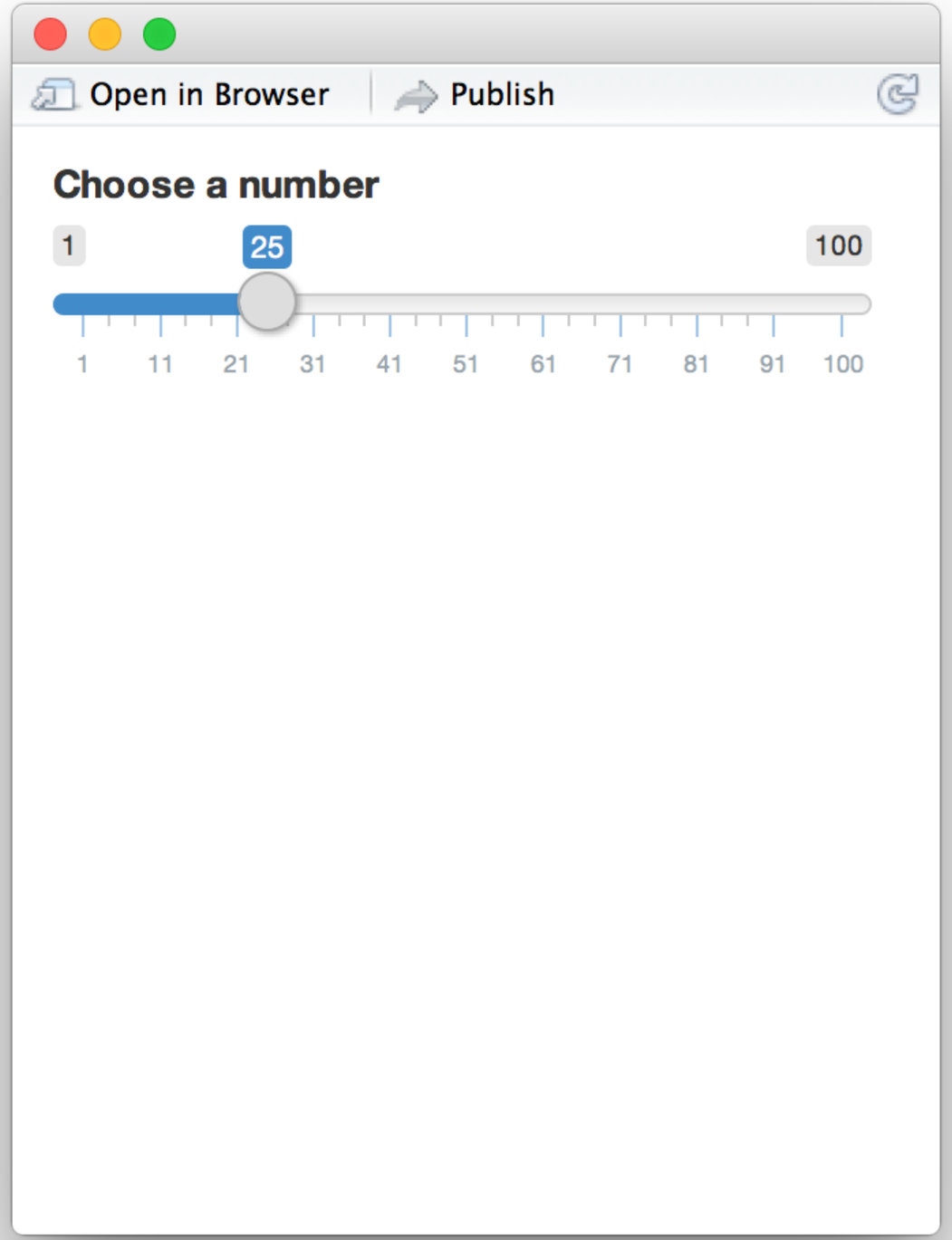
Add a slider input named **`"num"`** to your empty app (`exercise(20)`).

The Slider should begin at 25 and go from 1 to 100.

Use the help page at `?sliderInput` to determine which arguments to use:

1. inputId
2. label
3. ?
4. ?
5. ?

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100)
)

server <- function(input, output) {}

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



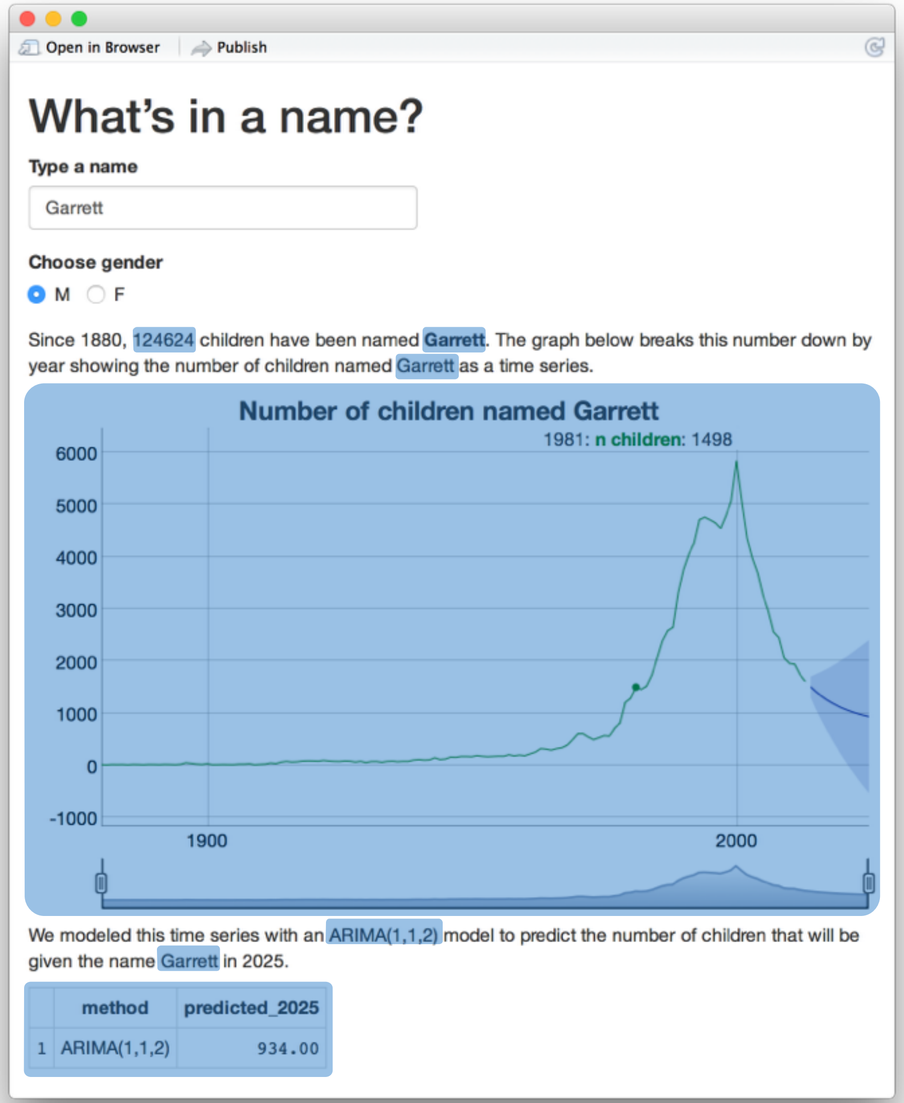
Outputs

2

Add elements to your app as arguments to `fluidPage()`

```
fluidPage(
  # input controls,
  # output controls,
  # layout objects,
  # custom HTML/CSS/JavaScript
)
```

Outputs - respond, if needed, when reactive values change



*Output()

To display output, add it to `fluidPage()` with an `*Output()` function

```
plotOutput(outputId = "barPlot")
```

- `plotOutput` — the type of output to display
- `plotOutput` — an *Output function
- `"barPlot"` — name of object

Function	Inserts
`dataTableOutput`	an interactive table
`htmlOutput`	raw HTML
`imageOutput`	image
`plotOutput`	plot
`tableOutput`	table
`textOutput`	text
`uiOutput`	a Shiny UI element
`verbatimTextOutput`	text

To display output, add it to the UI with an *Output function

```
library(shiny)
ui <- fluidPage(
  sliderInput("sl", "How loud should the
    guitar be?", min = 0, max = 11, val = 5),
  plotOutput("barPlot")
)

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

Comma between arguments

To display output, add it to the UI with an *Output function

```
library(shiny)
ui <- fluidPage(
  sliderInput("sl", "How loud should the
    guitar be?", min = 0, max = 11, val = 5),
  plotOutput("barPlot")
)

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

Does this create a plot?

Your turn

Return to the last exercise.

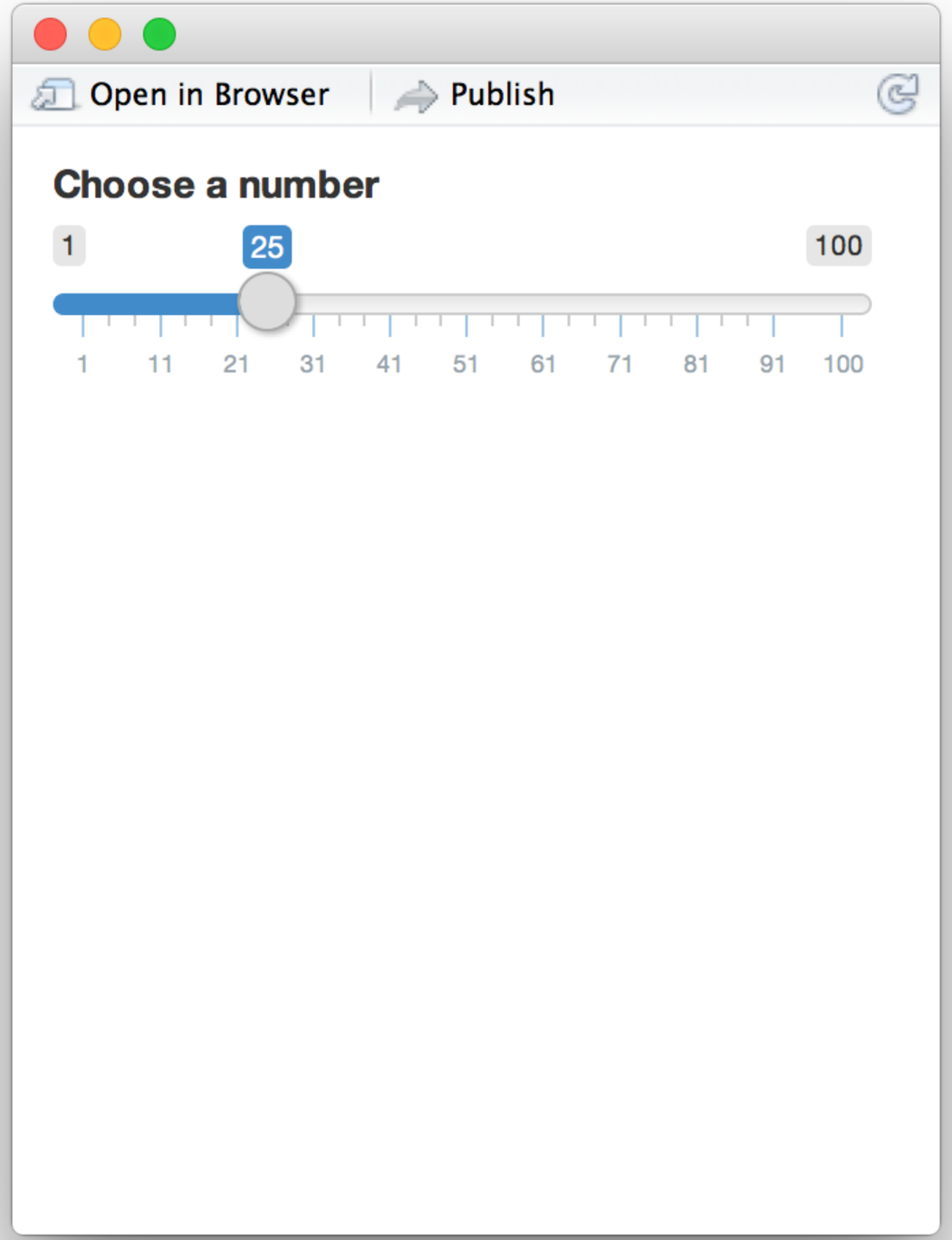
Place a plot named "hist" in your ui

Run the app. Does a plot appear?

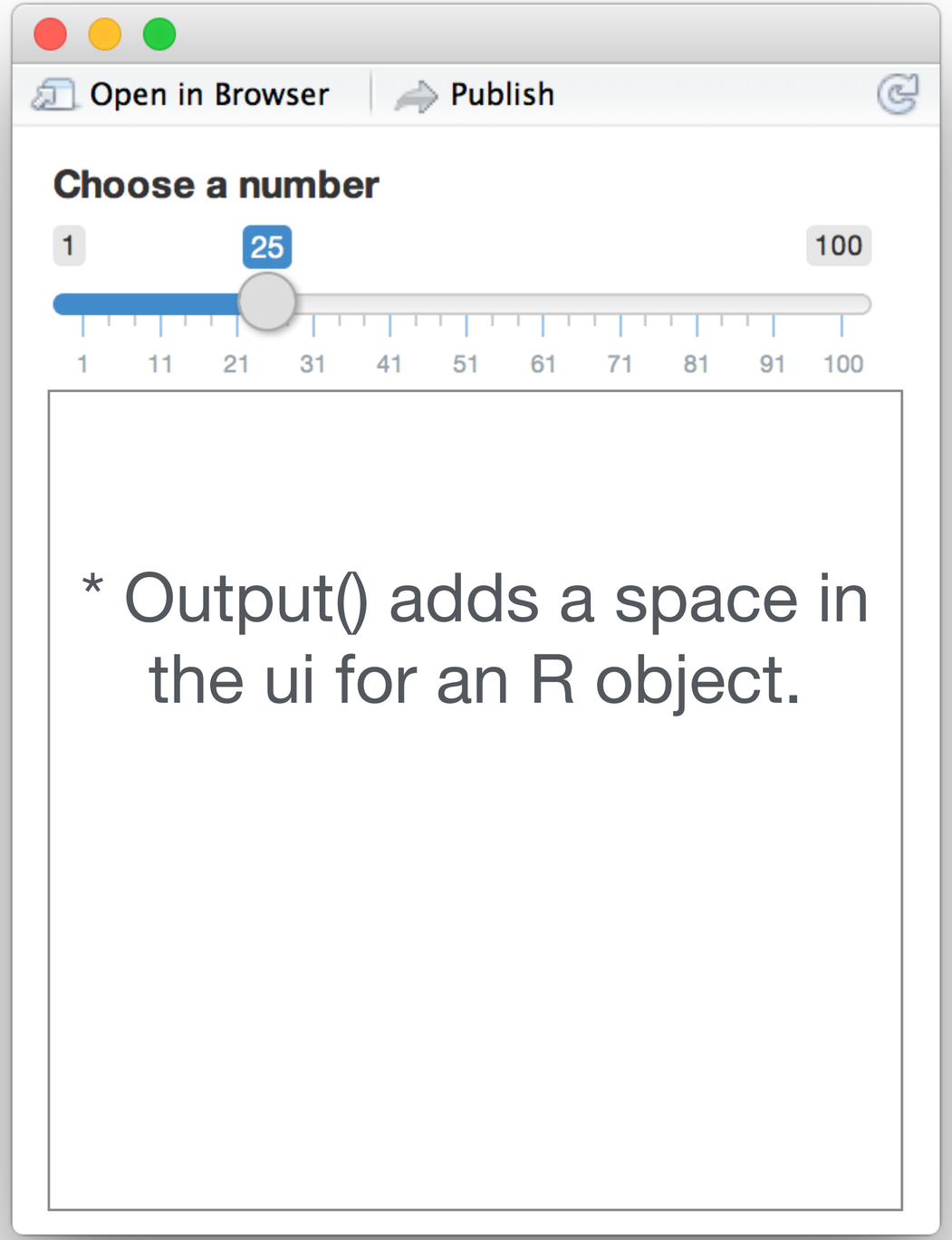
```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {}

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

Comma between arguments

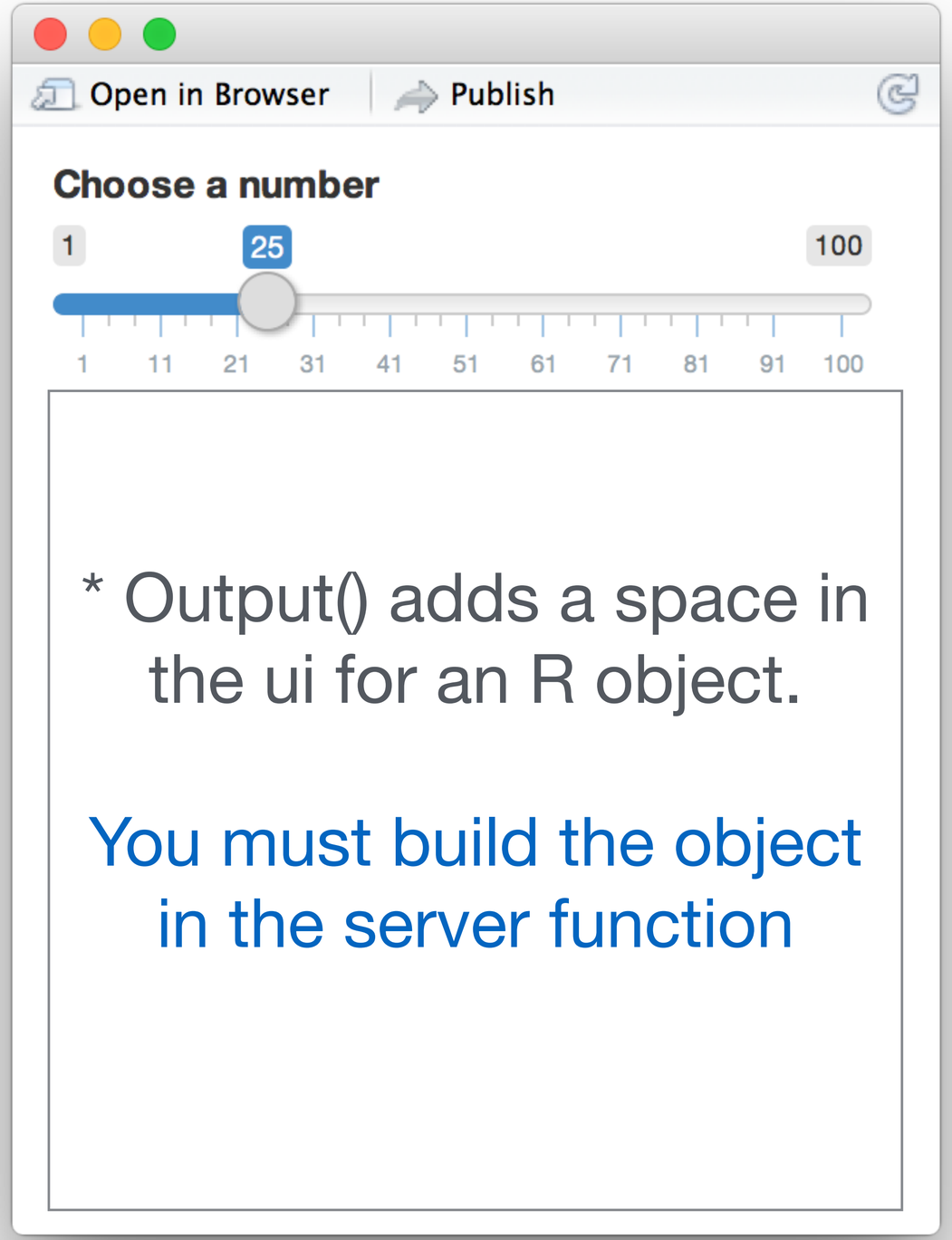
```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {}

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



```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {}

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

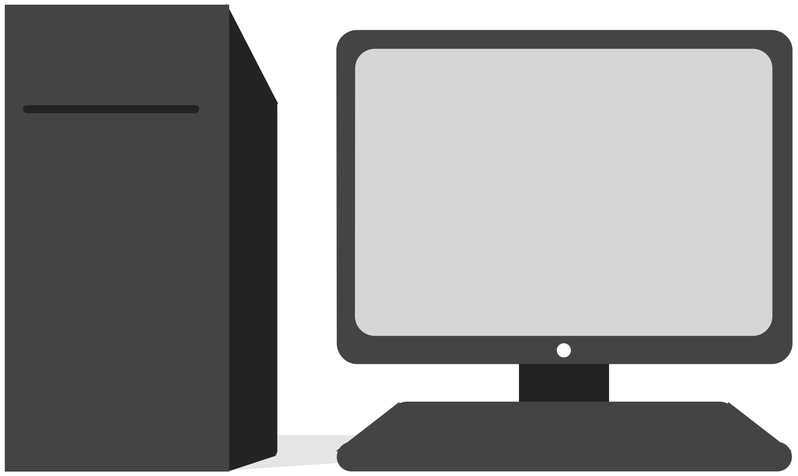


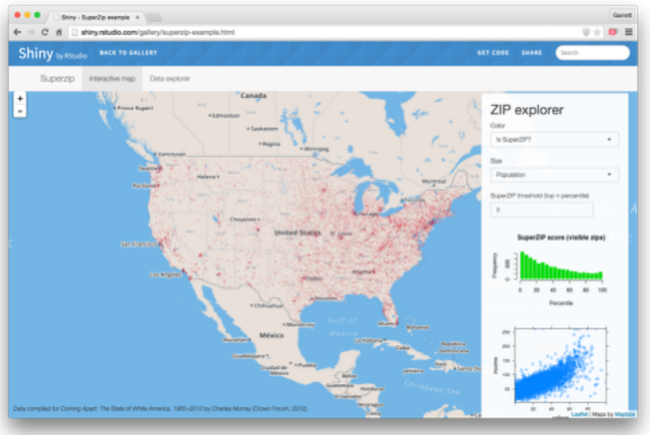
```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {}

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



server

Every Shiny app is maintained by a computer running R





1

The **server function** is a function that **you** must provide. Shiny calls it anytime a web browser connects to your app (i.e. starts a *session*). You set up a session's outputs, calculations, and actions in this function.

```
function(input, output, session) {


}
```

1

The server function has two required parameters (`input`, `output`) and one optional parameter (`session`). Shiny will provide these objects to you.

```
function(input, output, session) {


}
```

2

Use the function to build a list of outputs to display

```
function(input, output) {
  output$hist <- #?
}
```

```
output$hist
```

```
plotOutput("hist")
```

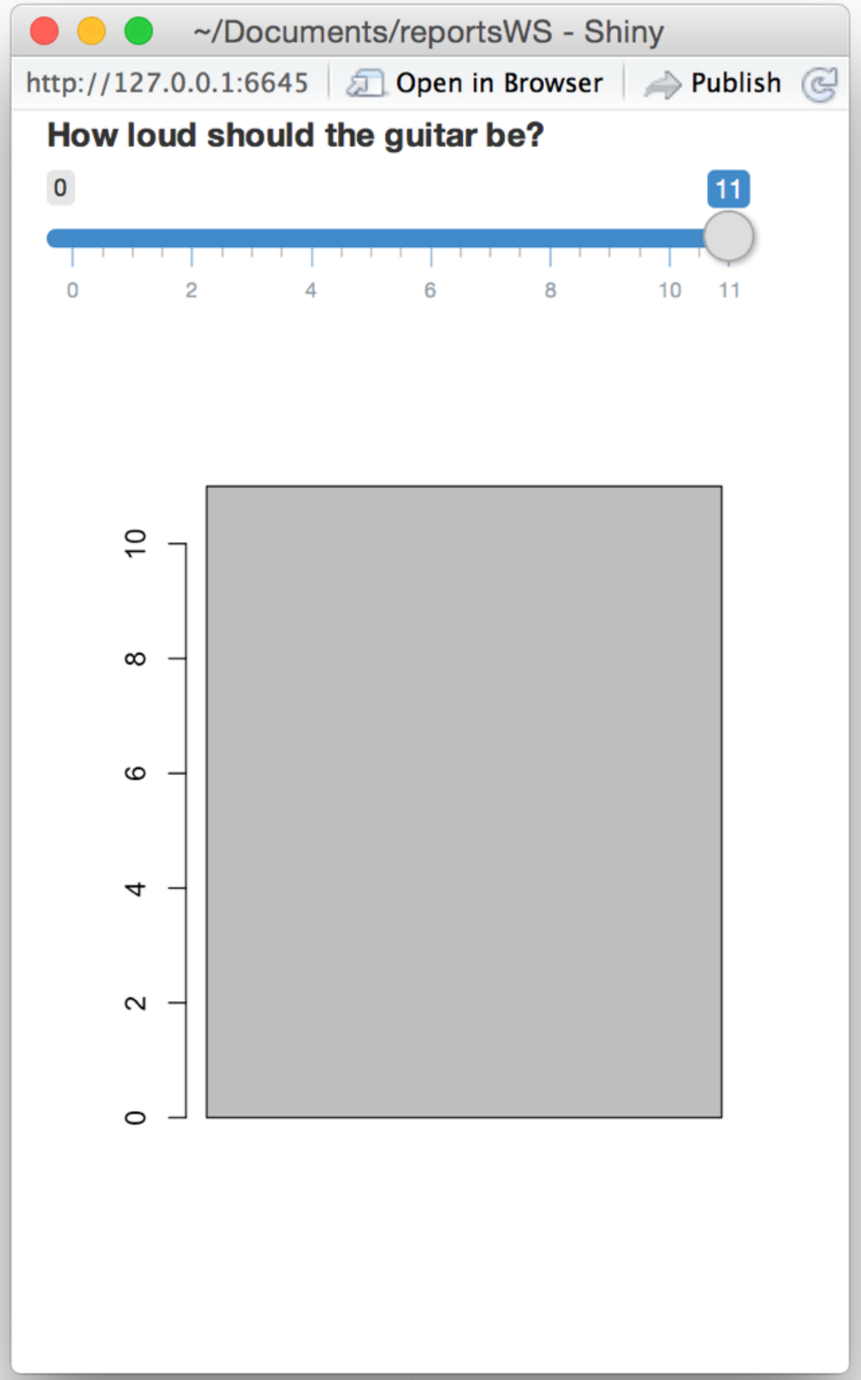
3

Build each object with a render* function

```
function(input, output) {
  output$hist <- renderPlot({
    # code that makes a plot
  })
})
```

```
library(shiny)
ui <- fluidPage(
  sliderInput("sl",
    "How loud should the guitar be?",
     min = 0. max = 11, val = 5),
  plotOutput("barPlot")
)

server <- function(input, output) {
  output$barPlot <-



}

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

```
library(shiny)
ui <- fluidPage(
  sliderInput("sl",
    "How loud should the guitar be?",
     min = 0, max = 11, val = 5),
  plotOutput("barPlot")
)

server <- function(input, output) {
  output$barPlot <- renderPlot({
    barplot(input$sl, ylim = c(0, 11))
  })
}

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



Your turn

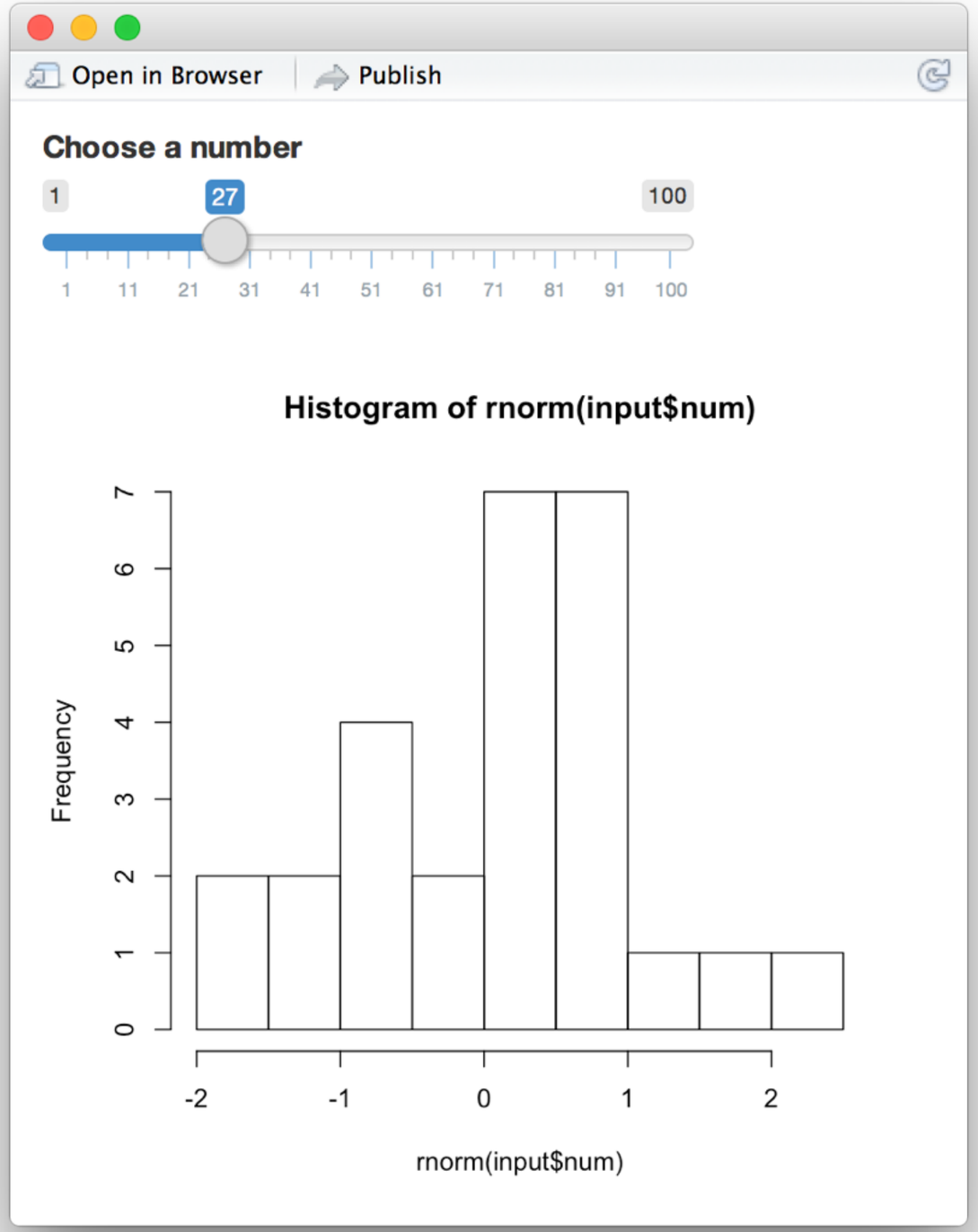
Return to the last exercise.

Render a histogram of `input$num` random normal values and and save it to `output$hist`, e.g.

```
hist(rnorm(input$num))
```

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <-



}

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

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

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



Reactive programming

Why reactivity?

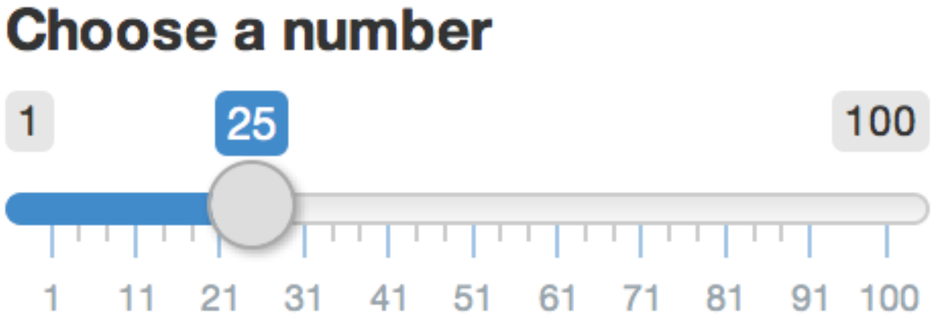
The goal is to respond to changes in user input, with the **minimal amount of computation** required to update the outputs: no more, no less.

Too much recomputation is **wasteful/slow**.

Too little recomputation gives us **incorrect results**.

Reactivity helps—a lot!

Syntax

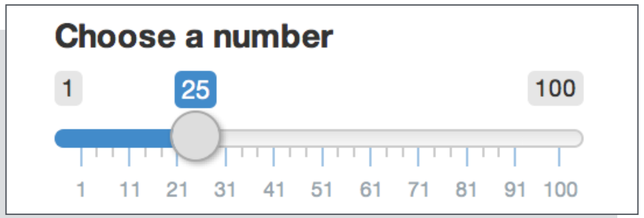



```
sliderInput(inputId = "num", label = "Choose a number", …)
```

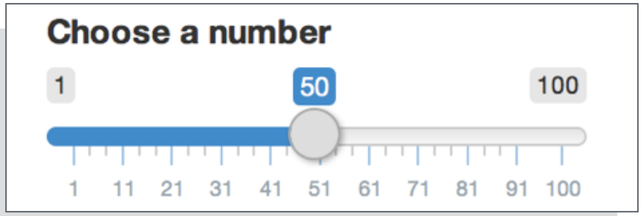
this input will provide a value saved as **input$num**

Input values

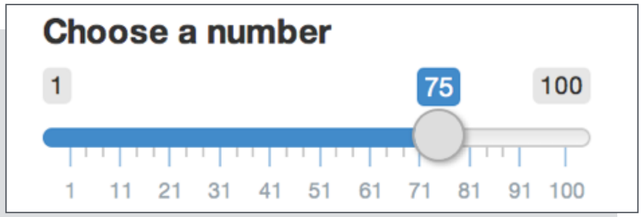
The input value changes whenever a user changes the input.




```
input$num = 25
```




```
input$num = 50
```

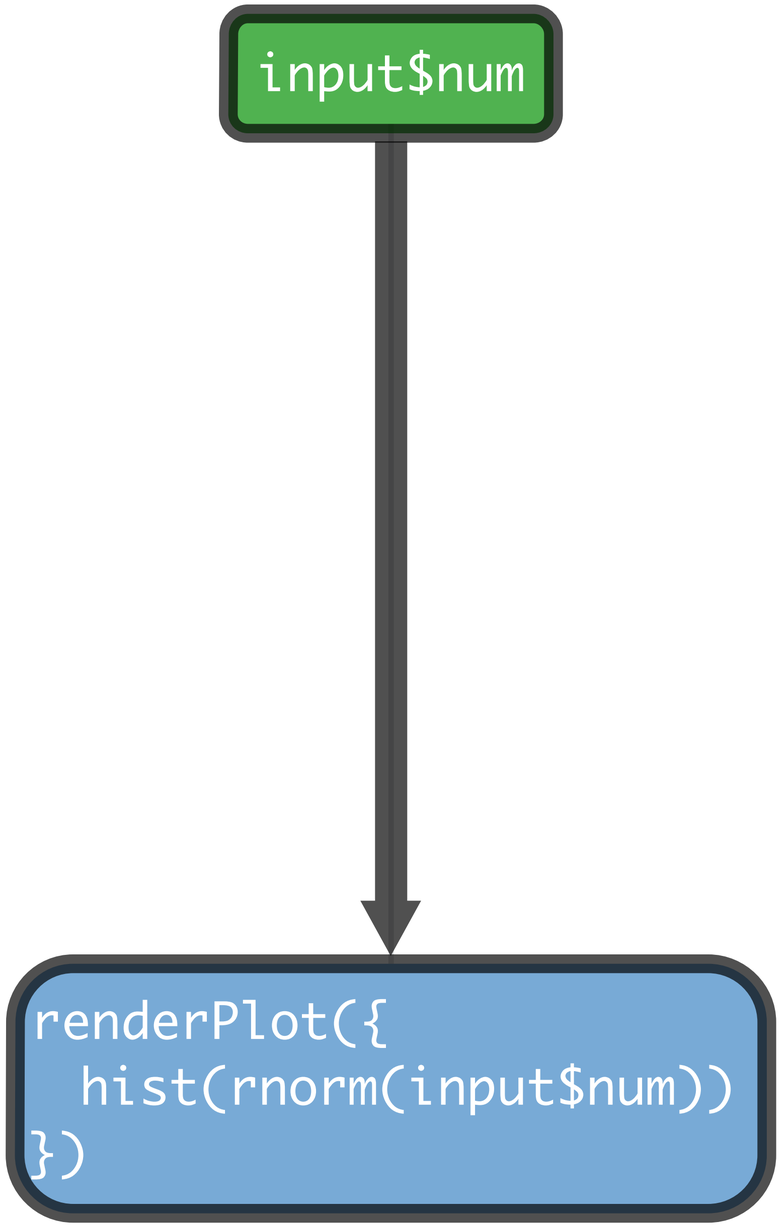


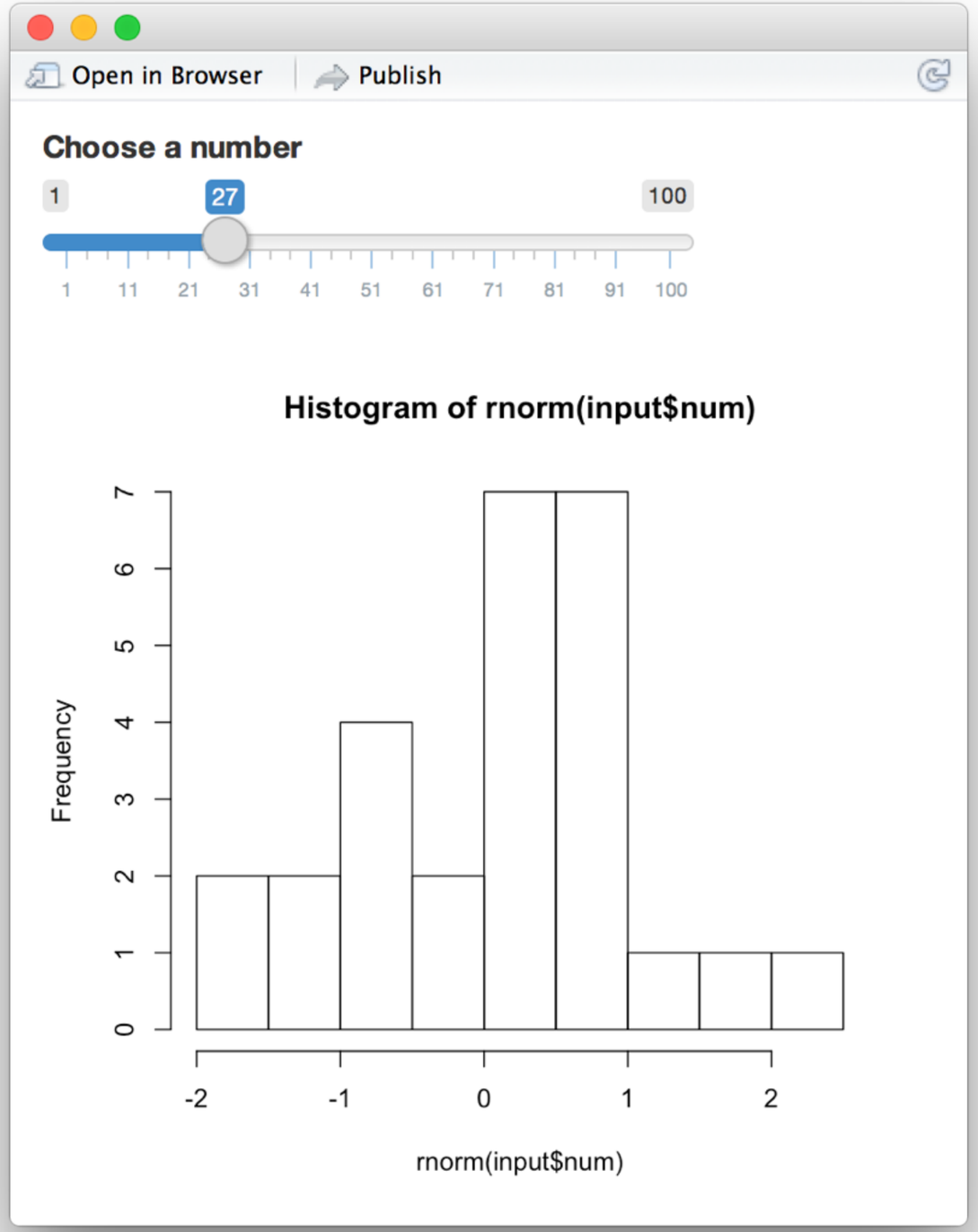

```
input$num = 75
```

Reactivity 101

Reactivity automatically occurs whenever you use an input value to render an output object

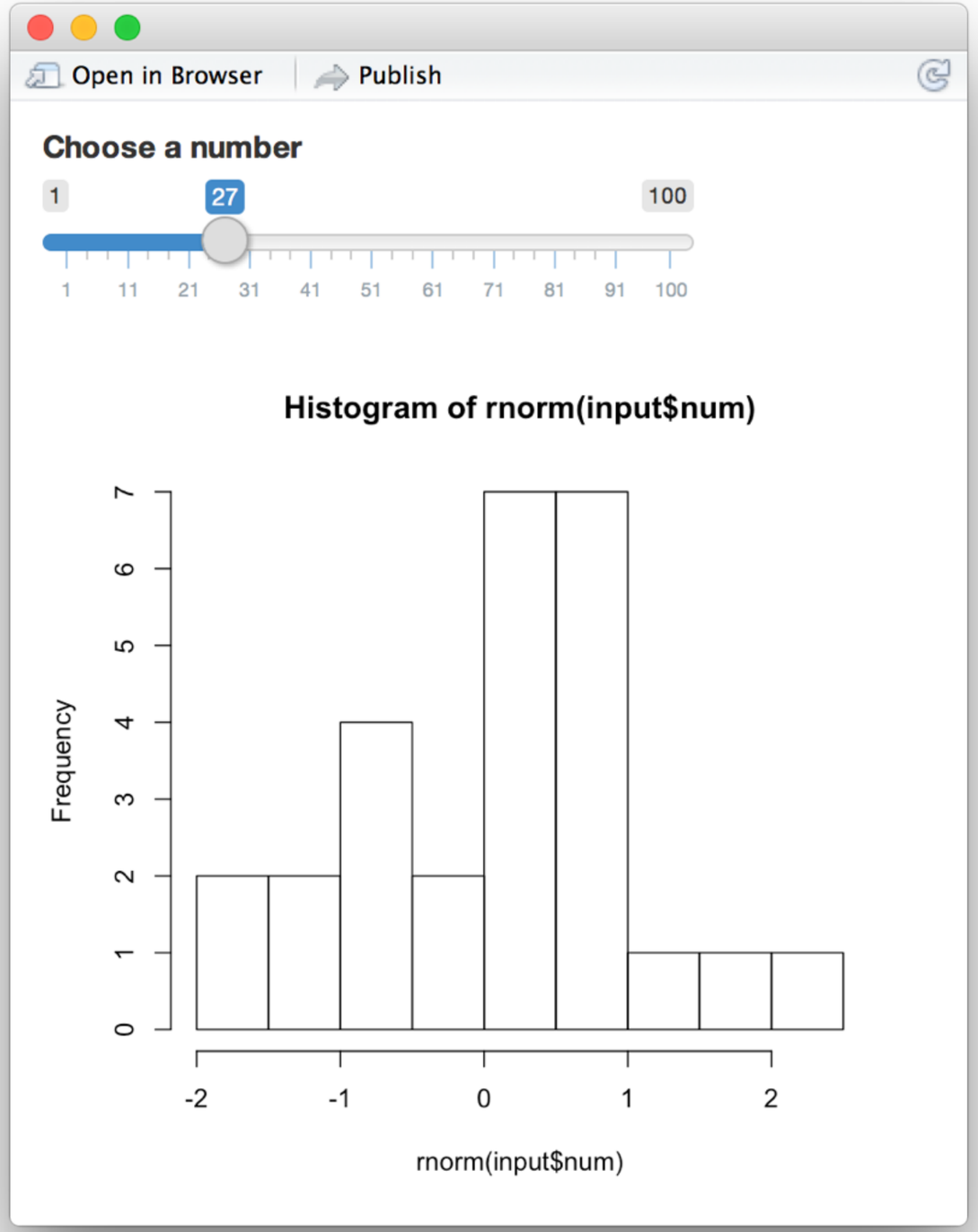
```
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
})
```

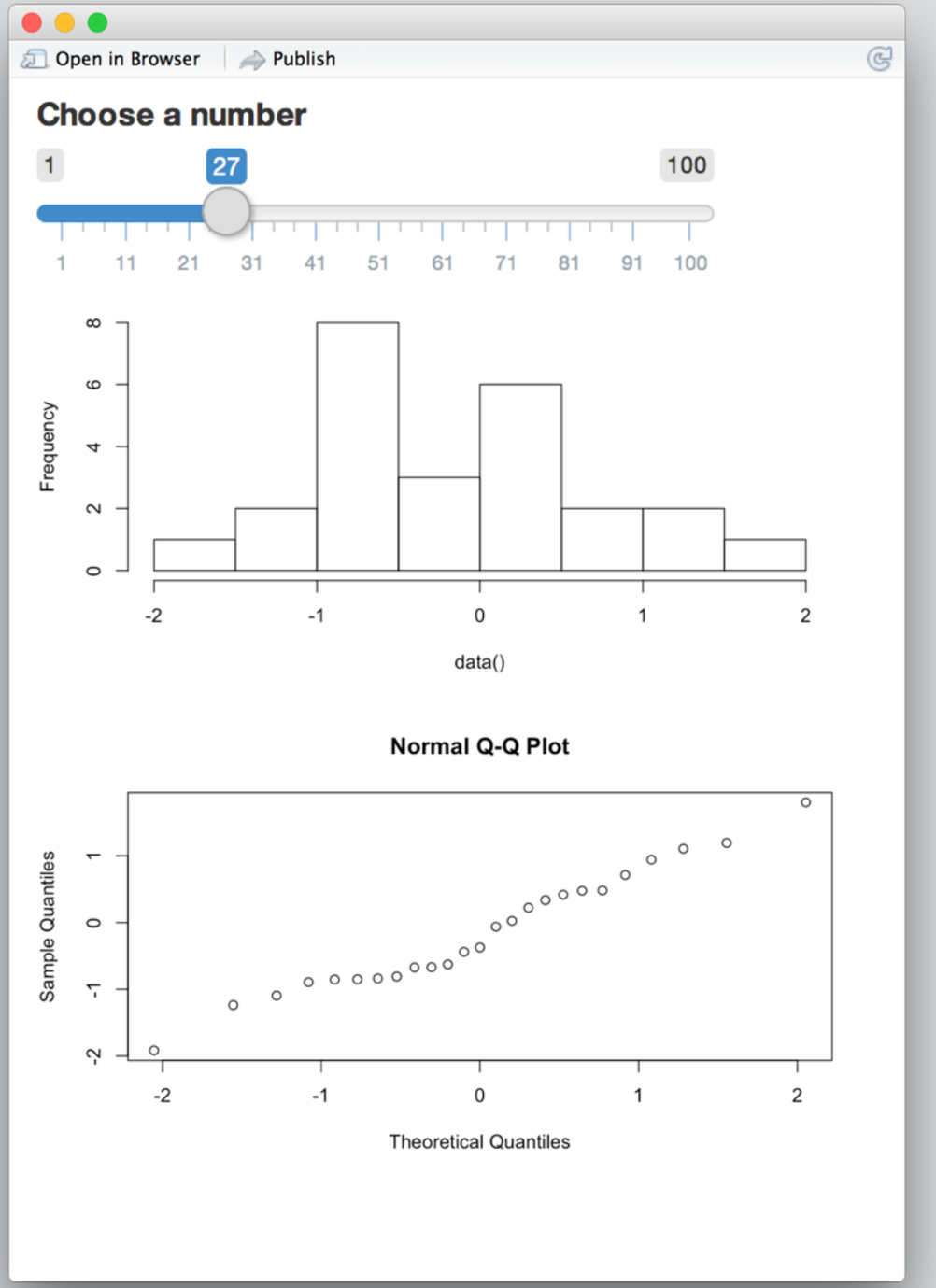




input$num

```
renderPlot({
  hist(rnorm(input$num))
})
```





Your turn

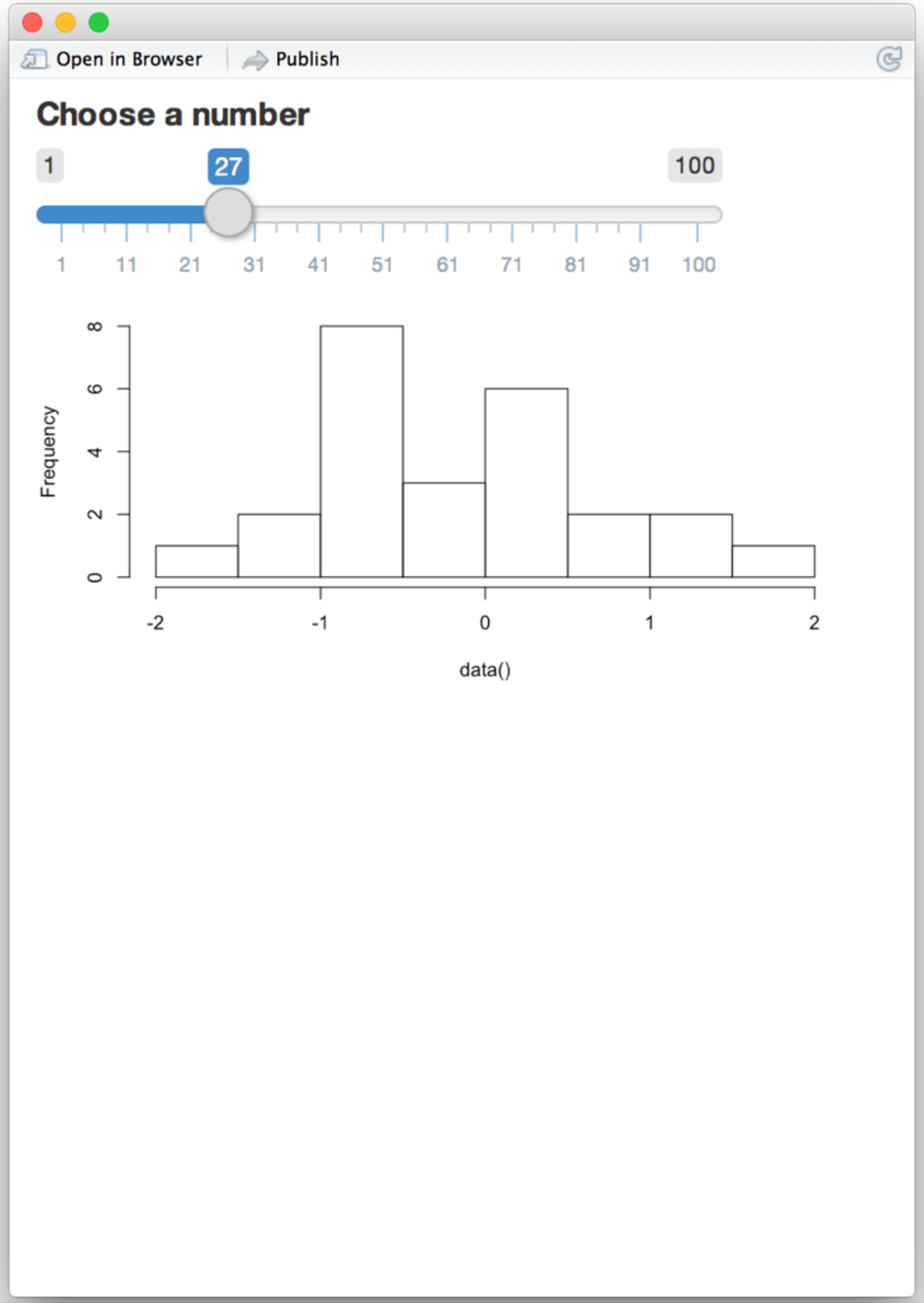
Add a second plot to your app.

It should contain a qqplot of `input $num` random normal values, e.g.

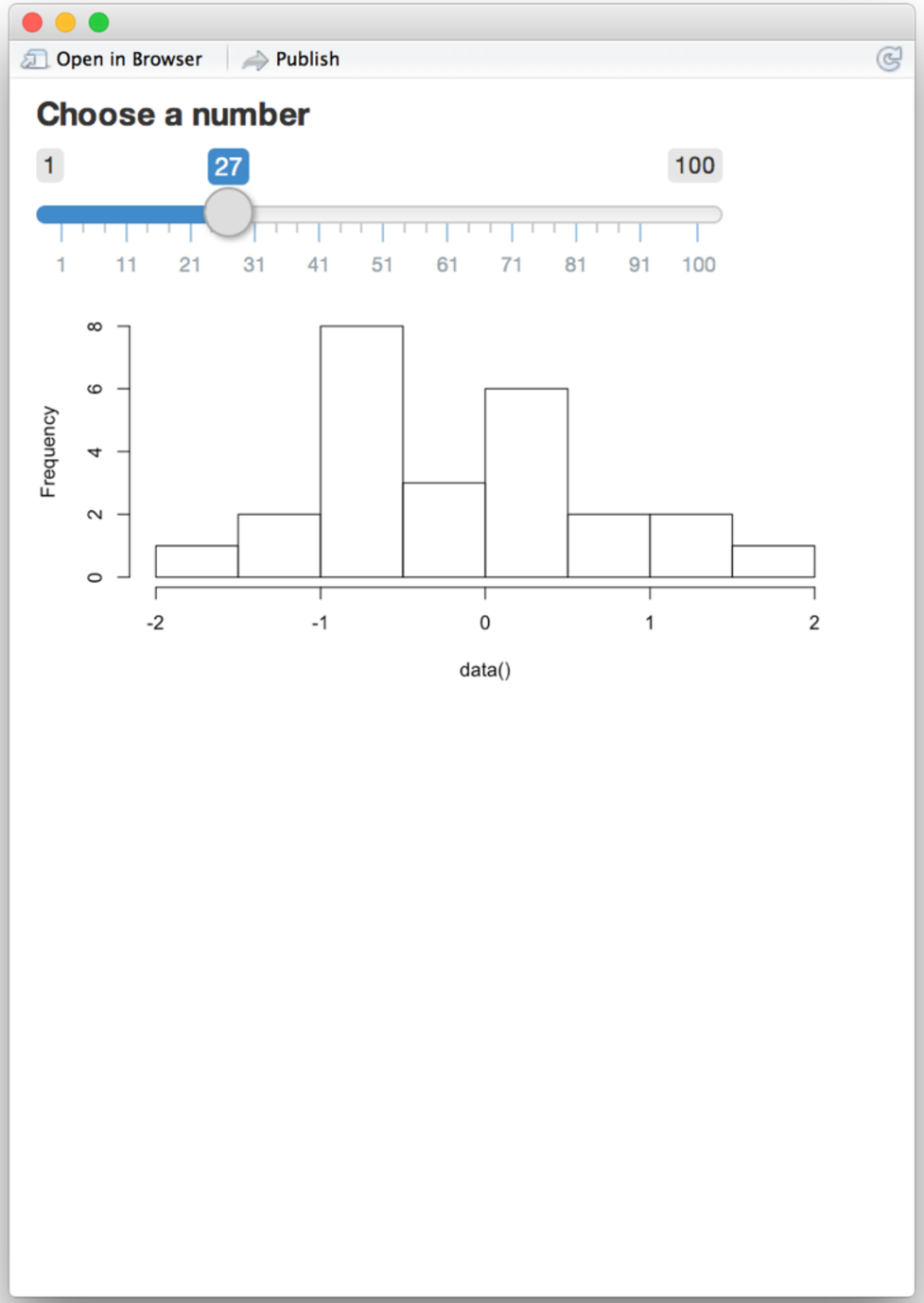
```
qqnorm(rnorm(input$num))
```

04:00

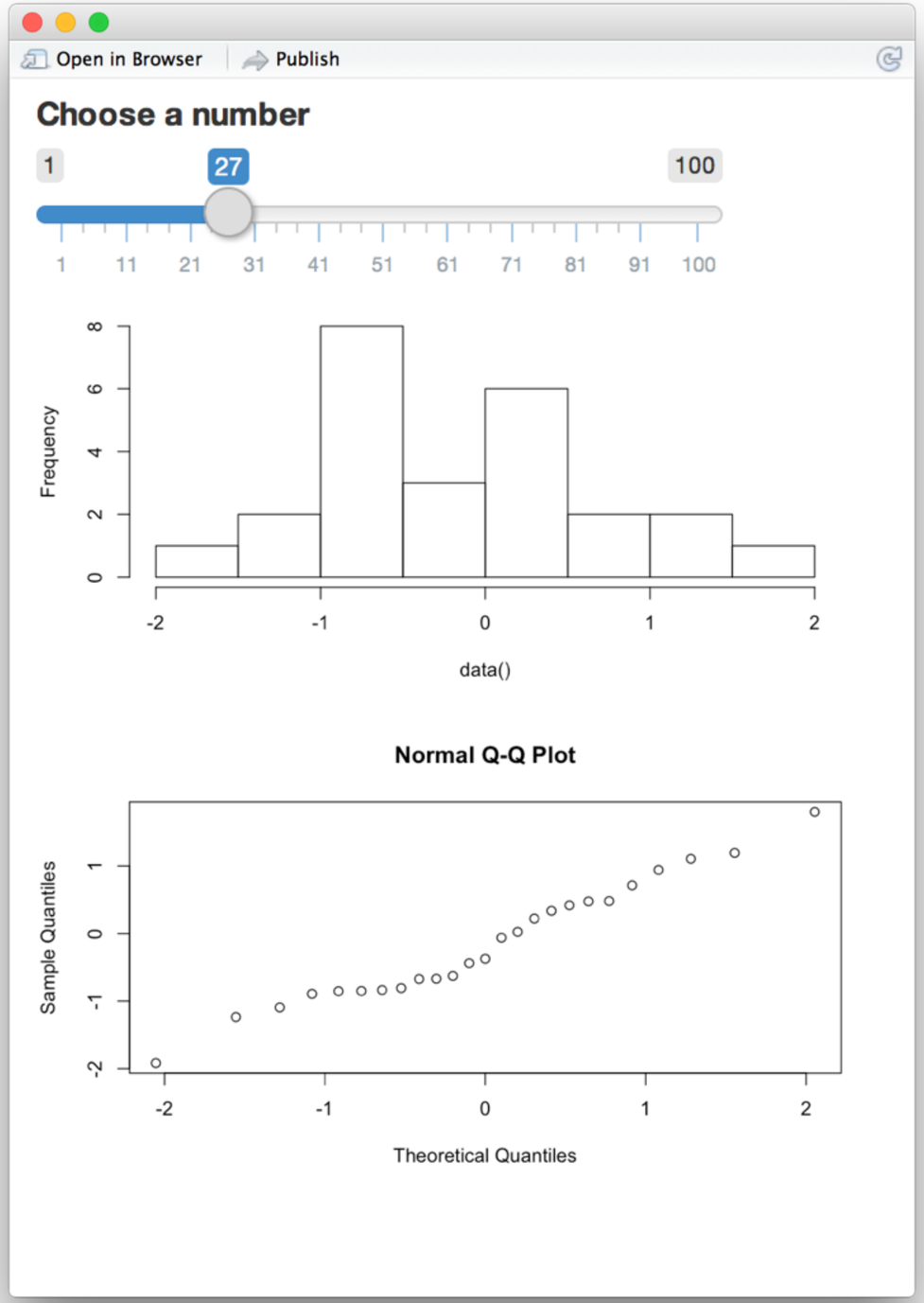
```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist"),
  plotOutput("qq")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })




}

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



```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist"),
  plotOutput("qq")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
  output$qq <-


}

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

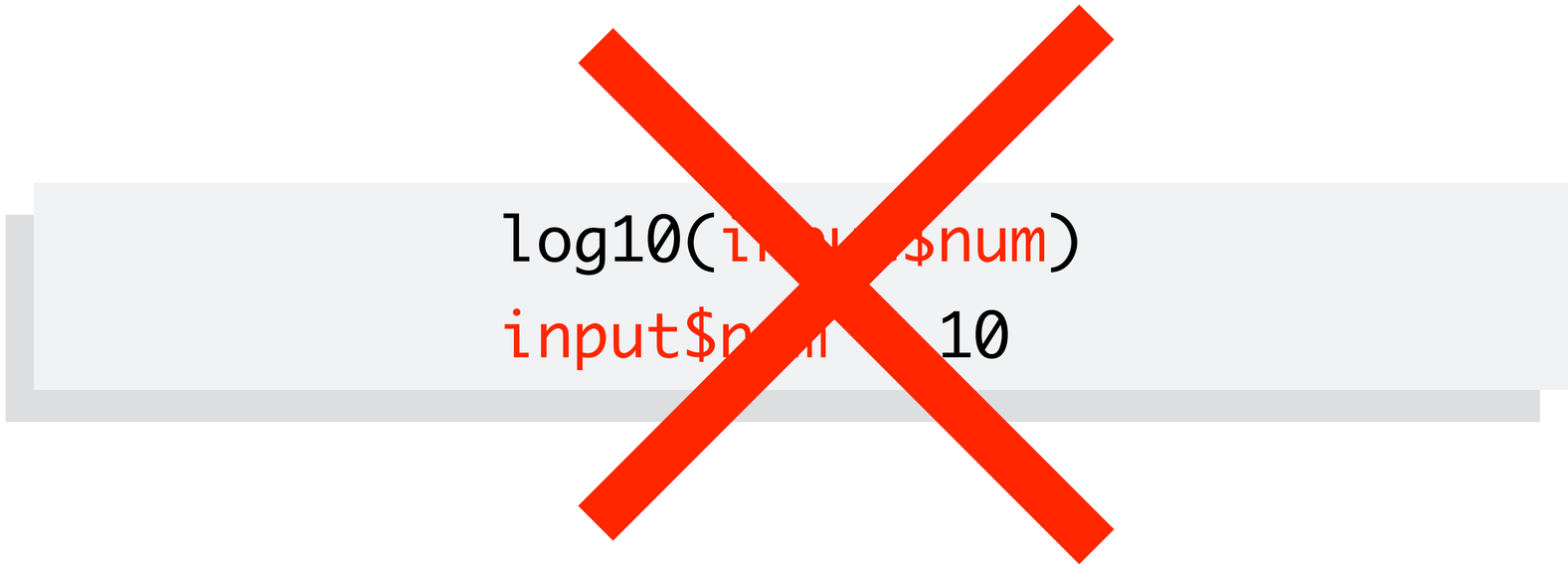


```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist"),
  plotOutput("qq")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
  output$qq <- renderPlot({
    qqnorm(rnorm(input$num))
  })
}

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



Reactivity 102

You can **only** call a reactive value from within a shiny function.



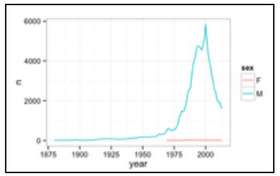
Recap: Shiny apps

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

Build an app with a **ui object**, a **server function**, and **shinyApp()**

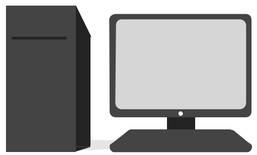
The ui object lays out what your user will see

Create reactive inputs with an ***Input()** function



Display reactive results with an ***Output()** function

Recap: Shiny apps

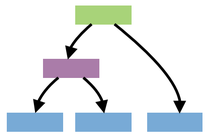


Give the server a set of instructions with the server function

`output$qq <-` Save reactive output to **output$**

```
renderPlot({
  hist(rnorm(input$num))
})
```

Build reactive output with a **render*()** function



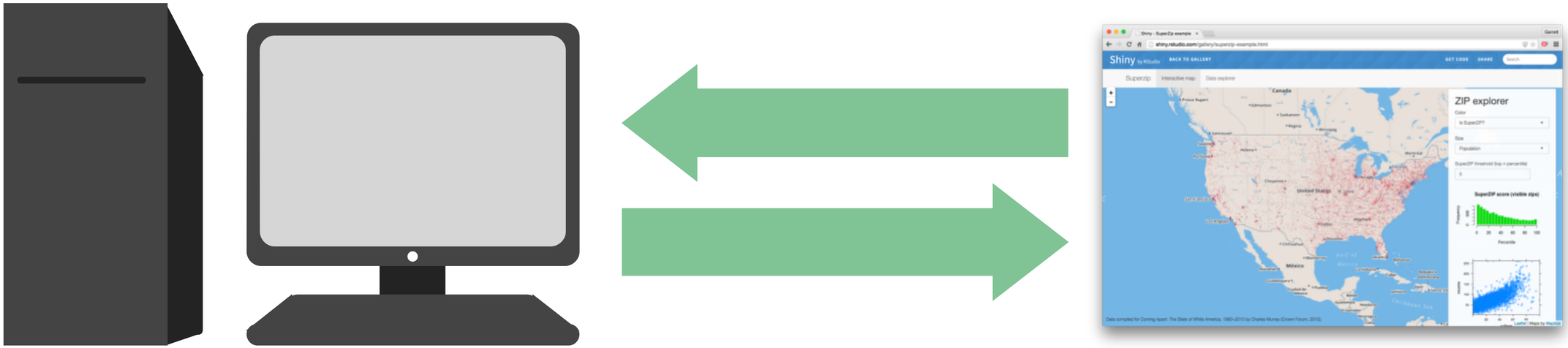
Create reactivity by chaining **Inputs** to (**reactive expressions** to) **rendered displays**

Reactive toolkit

(5 indispensible functions)

Tip

These functions give the server instructions that explain **what** code to run and **when** to run it.



Guiding questions

1 What code will the server run?

- some code that uses **reactive values**

2 When will it run it?

- when a **reactive value** changes

render*()

render*()

Builds reactive output to display in UI.

The return value must be assigned to `output$outputId`.

```
renderPlot(  { hist(rnorm(input$num)) })
```

a shiny function
(can handle reactive values)

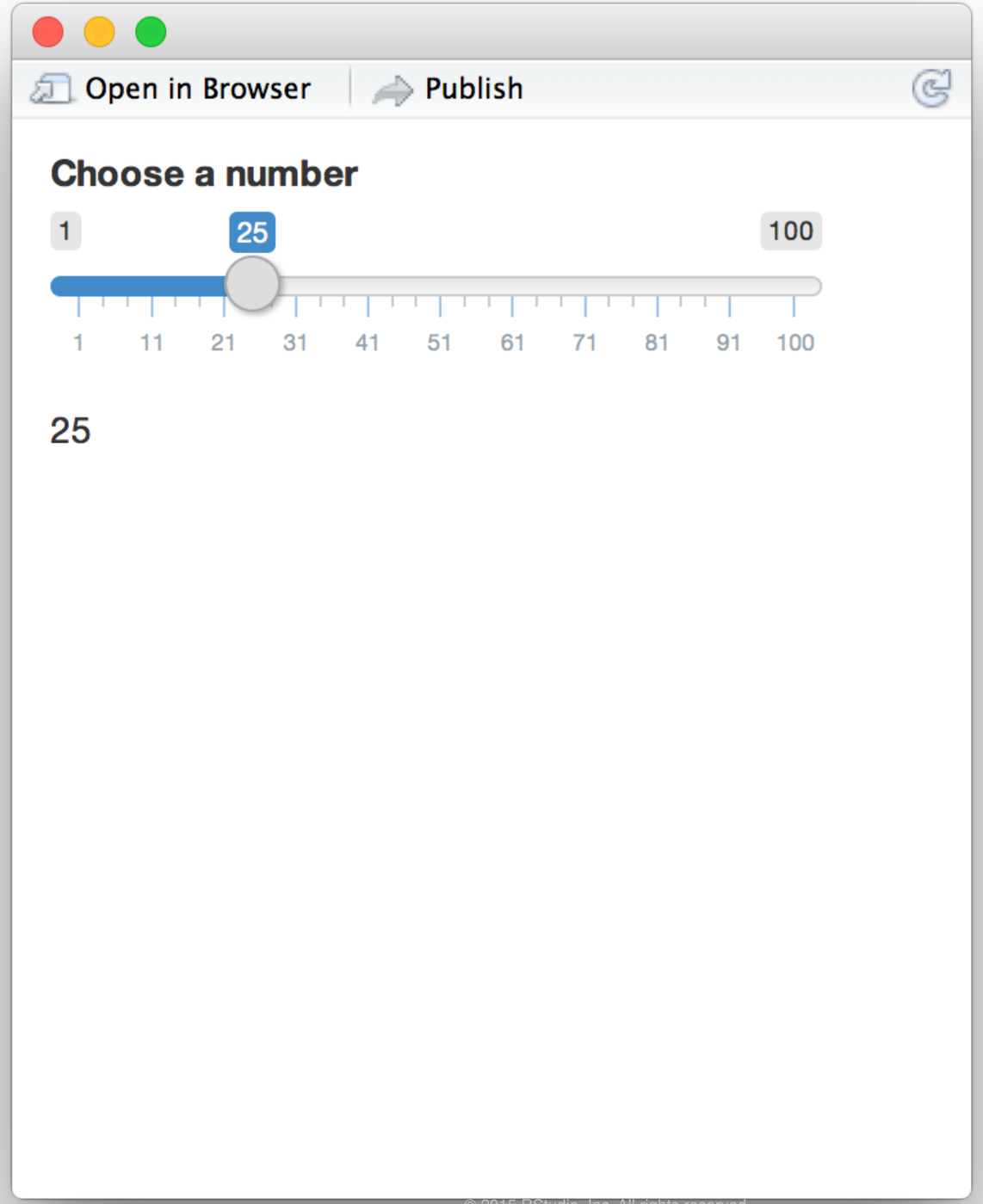
reruns whenever a reactive value in the block changes

code block to run

Use the function that creates the type of output you wish to make.

function	creates
`renderDataTable()`	An interactive table (from a data frame, matrix, or other table-like structure)
`renderImage()`	An image (saved as a link to a source file)
`renderPlot()`	A plot
`renderPrint()`	A code block of printed output
`renderTable()`	A table (from a data frame, matrix, or other table-like structure)
`renderText()`	A character string
`renderUI()`	a Shiny UI element

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  textOutput("n")
)

server <- function(input, output) {
  output$n <- renderText({
    input$num
  })
}

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



reactive()

reactive()

Builds a new reactive value (reactive expression)

```
data <- reactive(   { rnorm(input$num) })
```

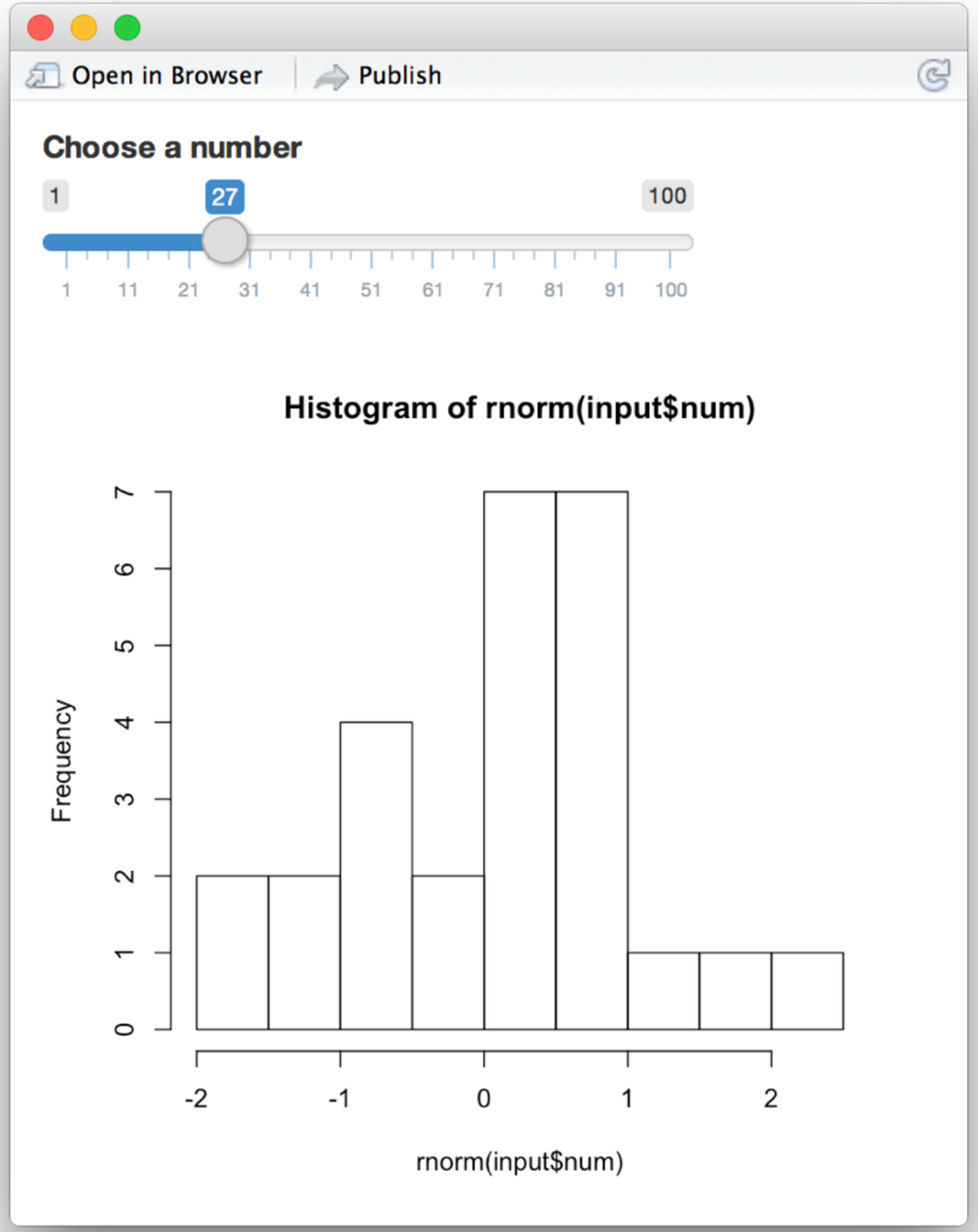
A function to call

a shiny function
(can handle reactive values)

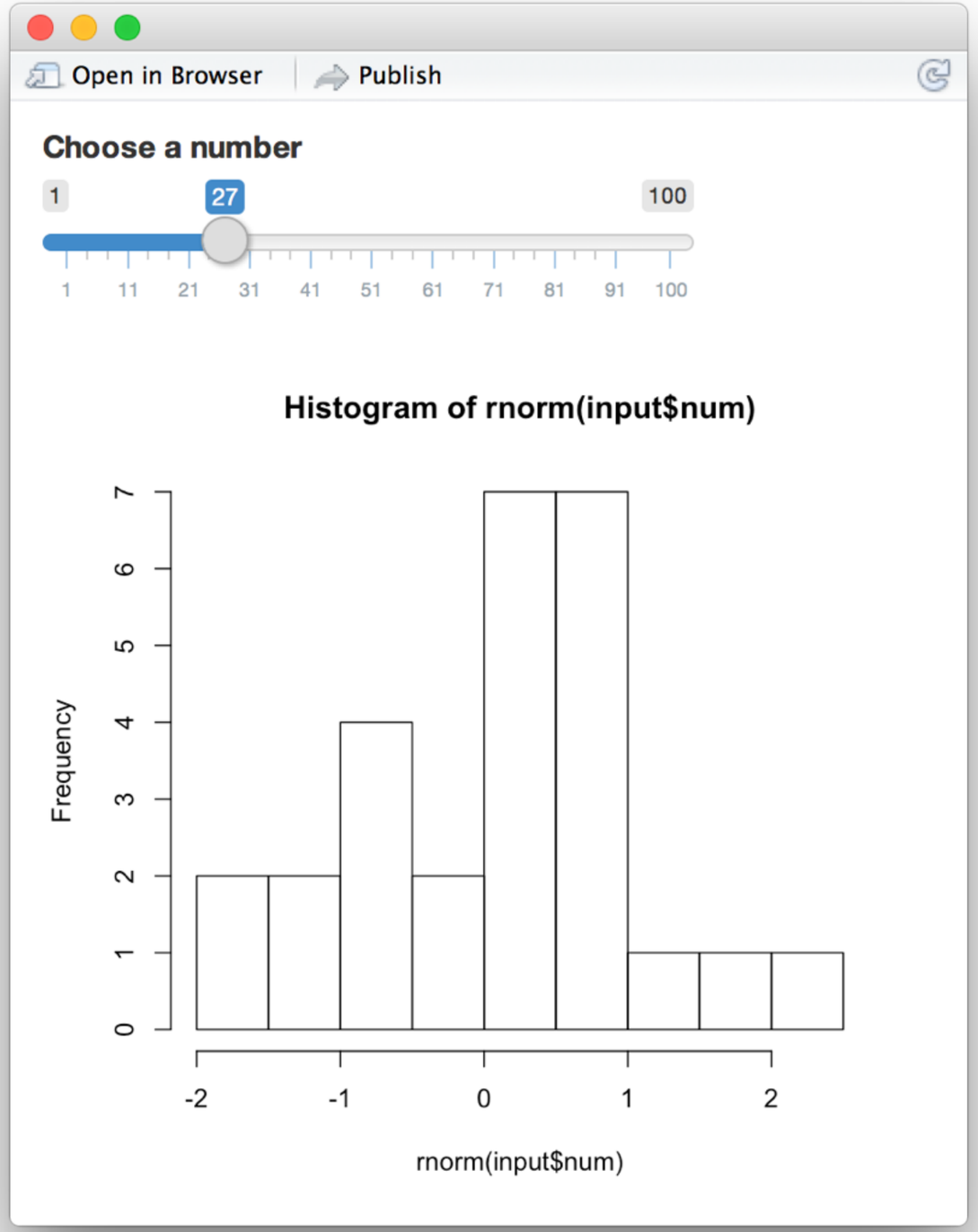
Responds to changes in any reactive value in block

code block to run when data() is called

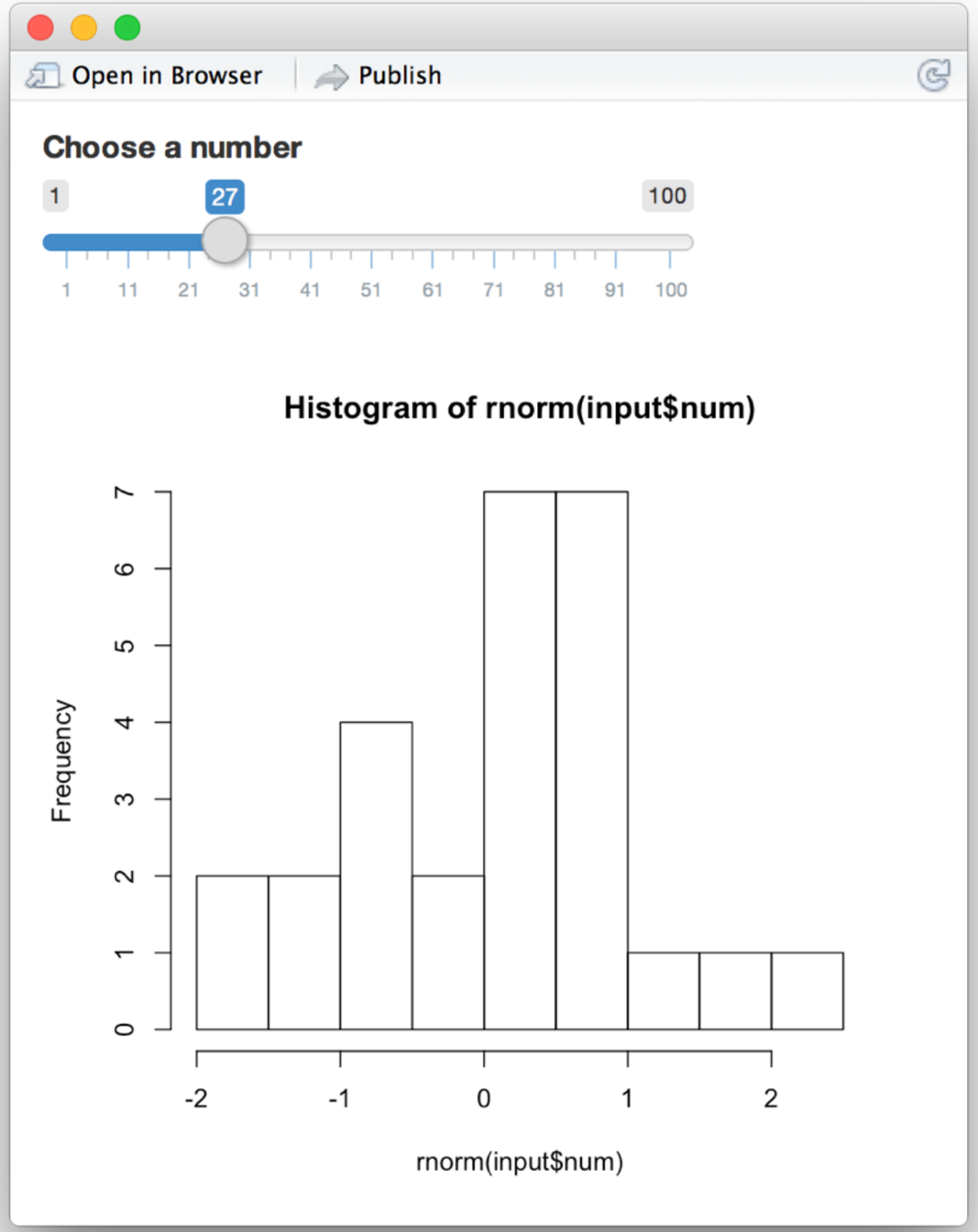
```
library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {

  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

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

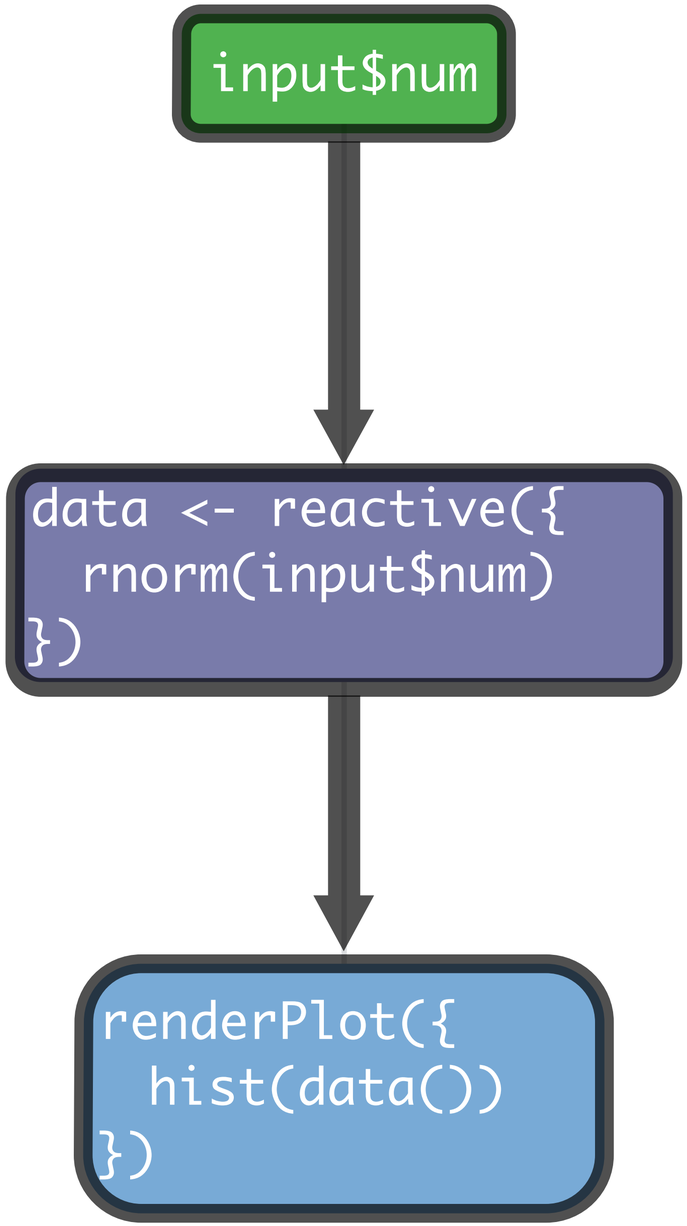


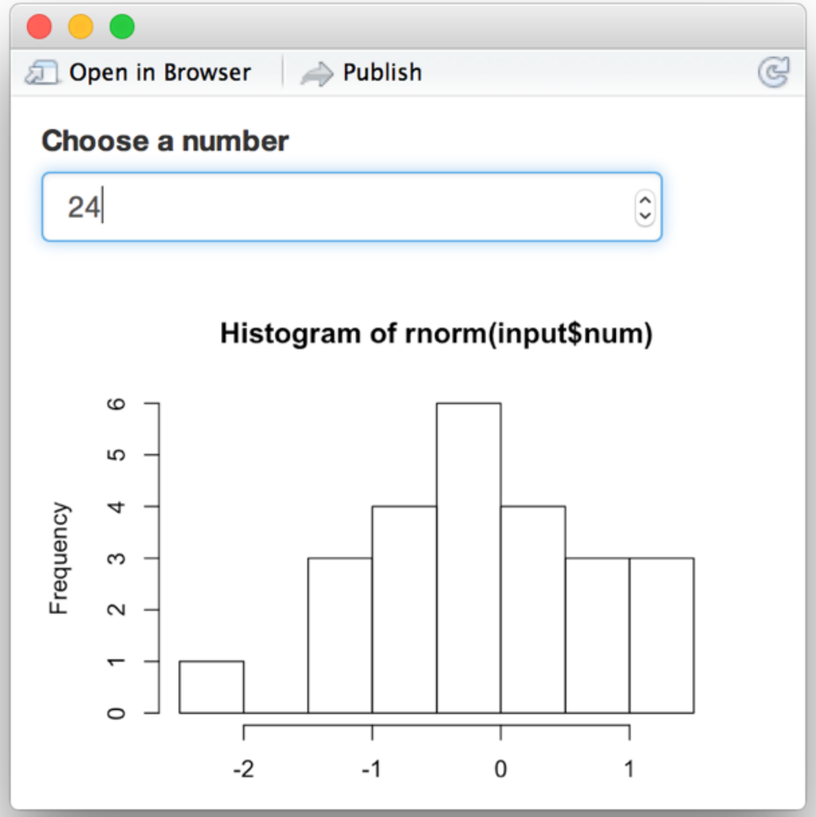
```
library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  data <- reactive({rnorm(input$num)})
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

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



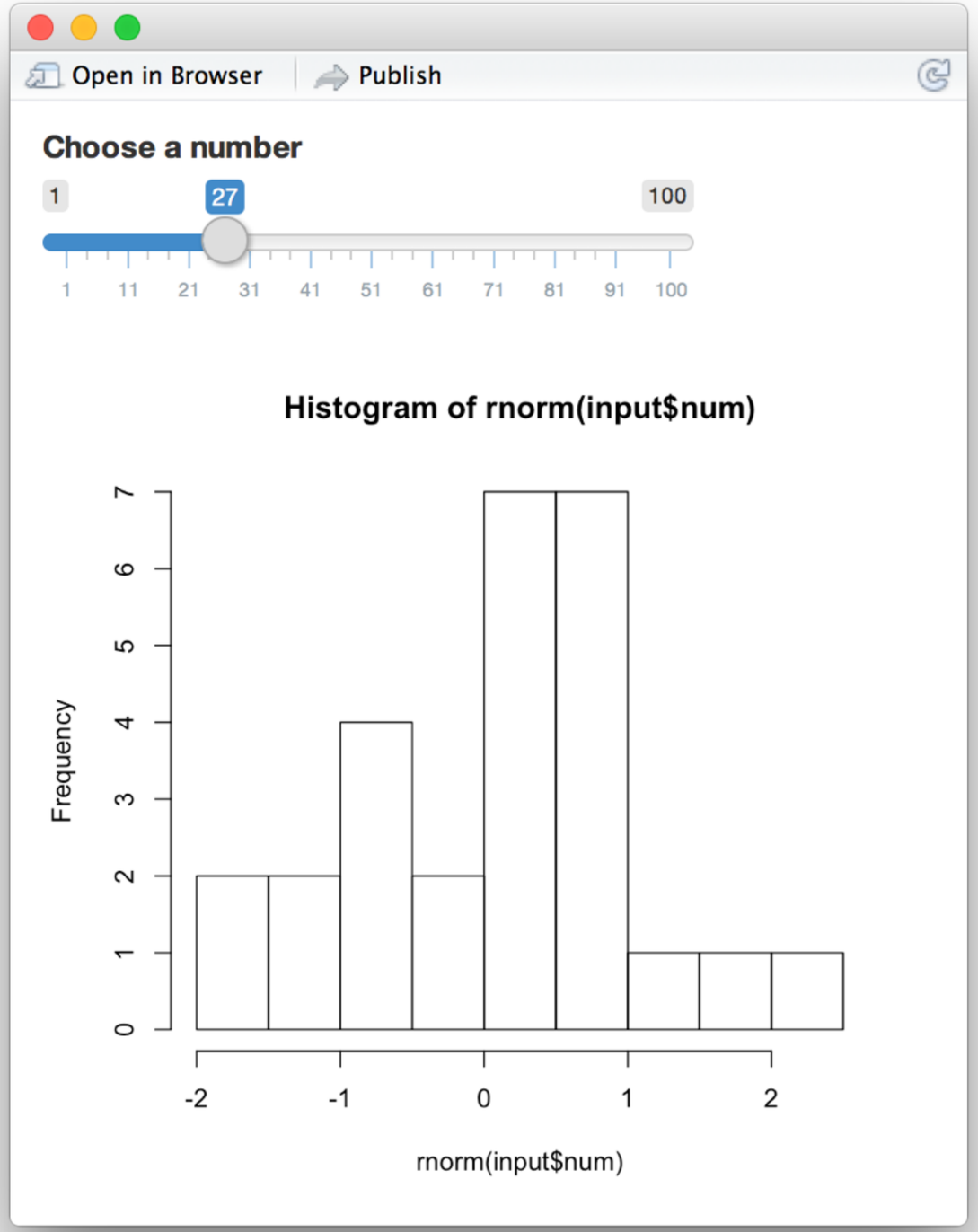
```
library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  data <- reactive({rnorm(input$num)})
  output$hist <- renderPlot({
    hist(data())
  })
}

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



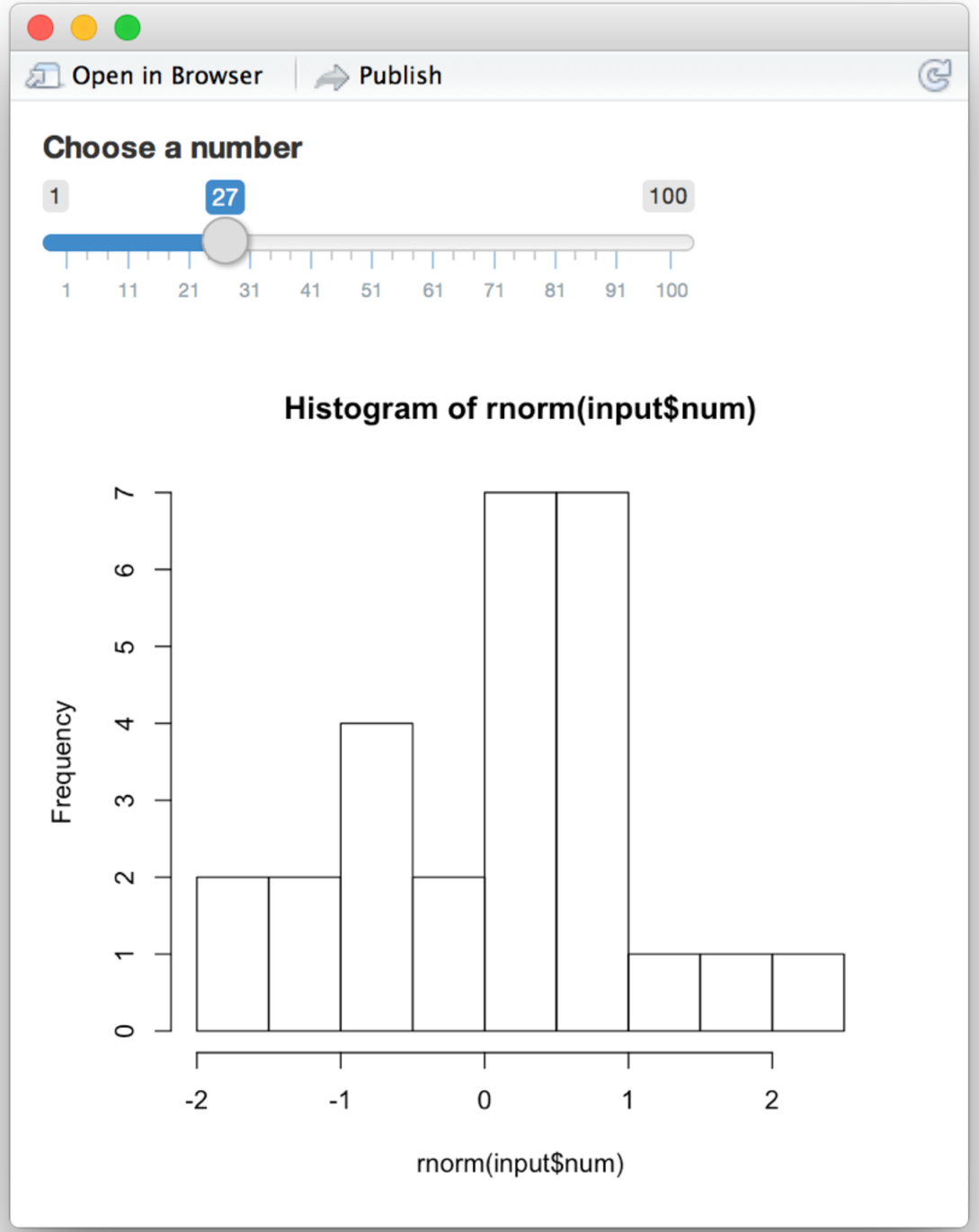




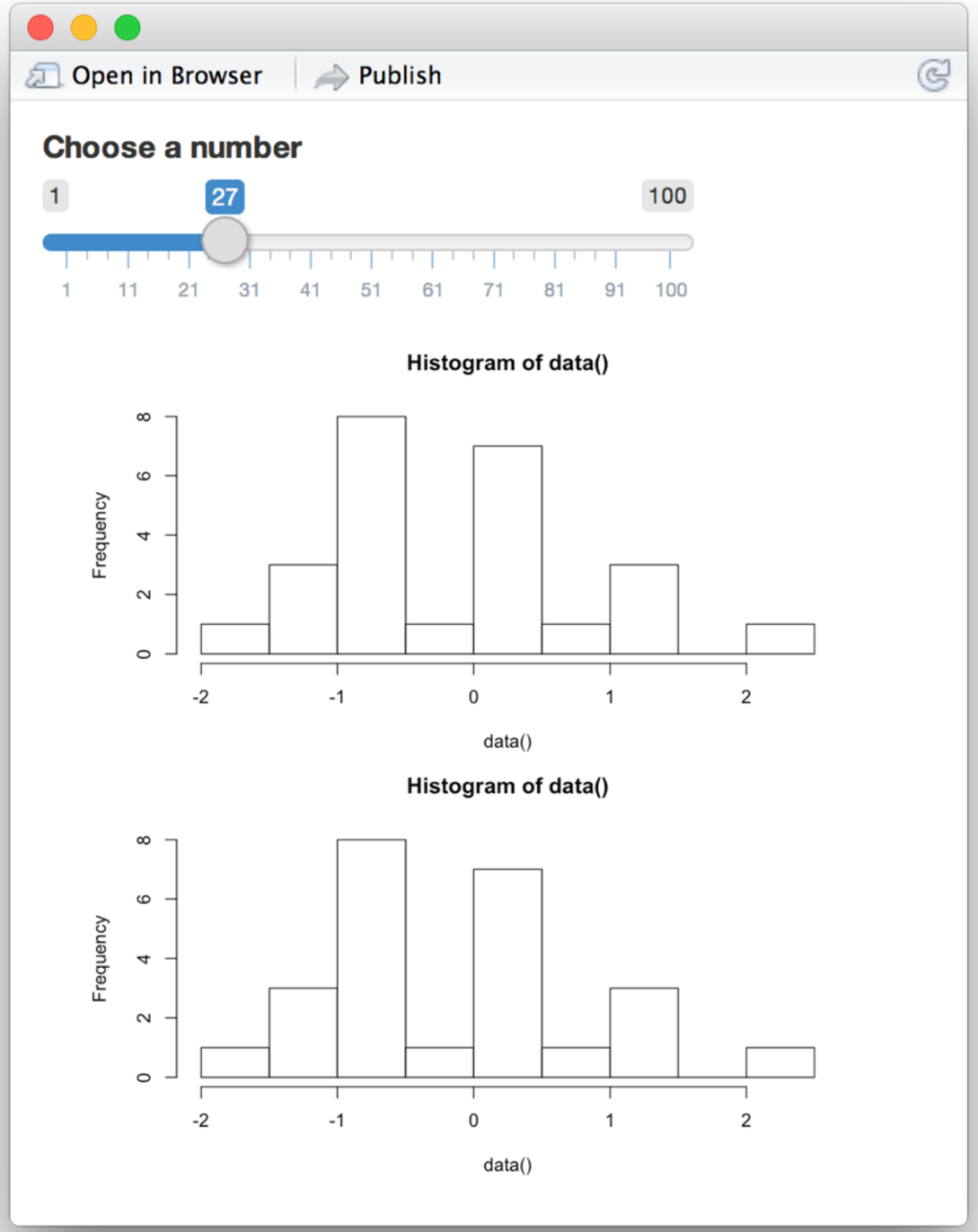
```
library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")

)
server <- function(input, output) {
  data <- reactive(rnorm(input$num))
  output$hist <- renderPlot({
    hist(data())
  })



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

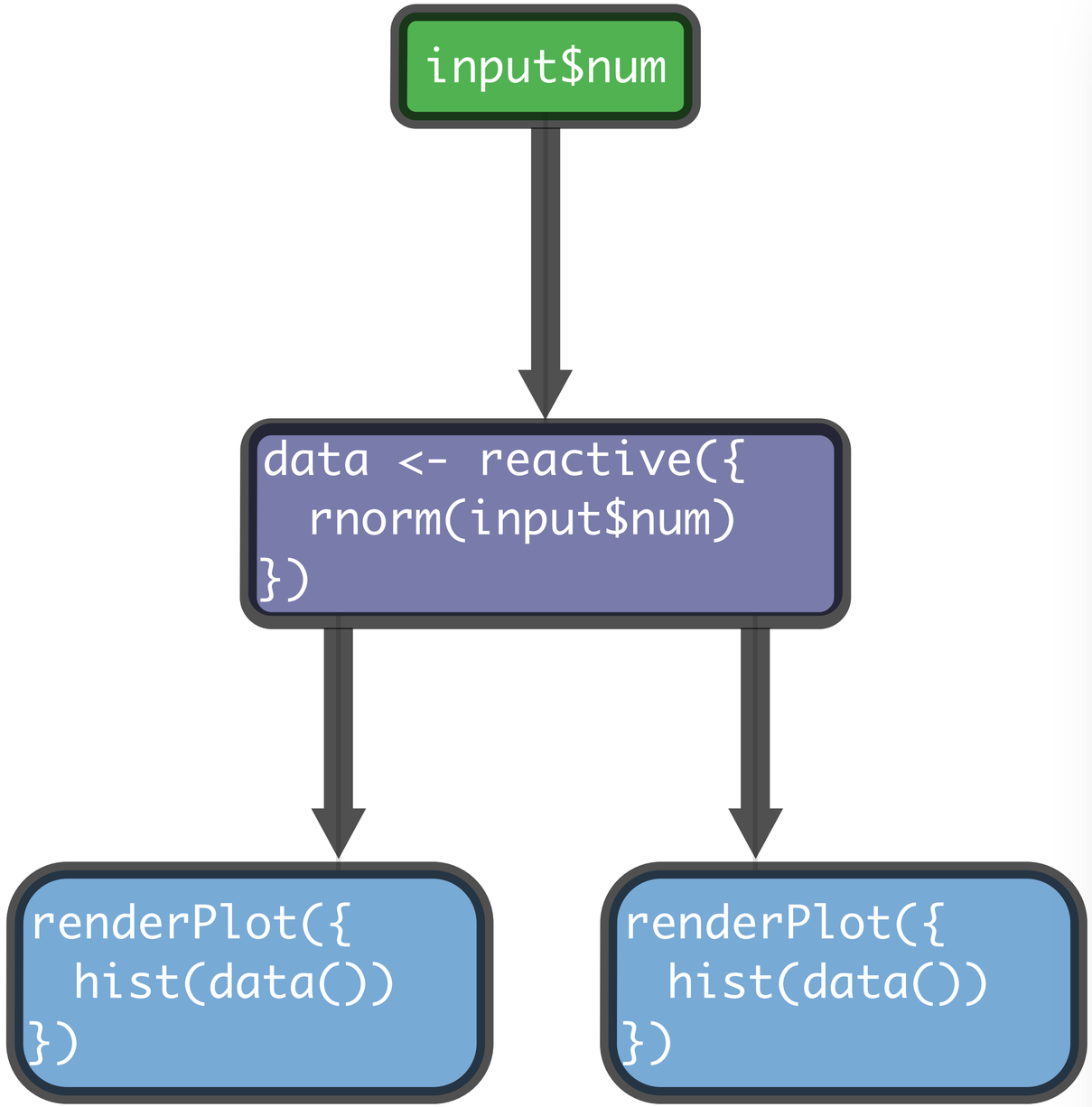


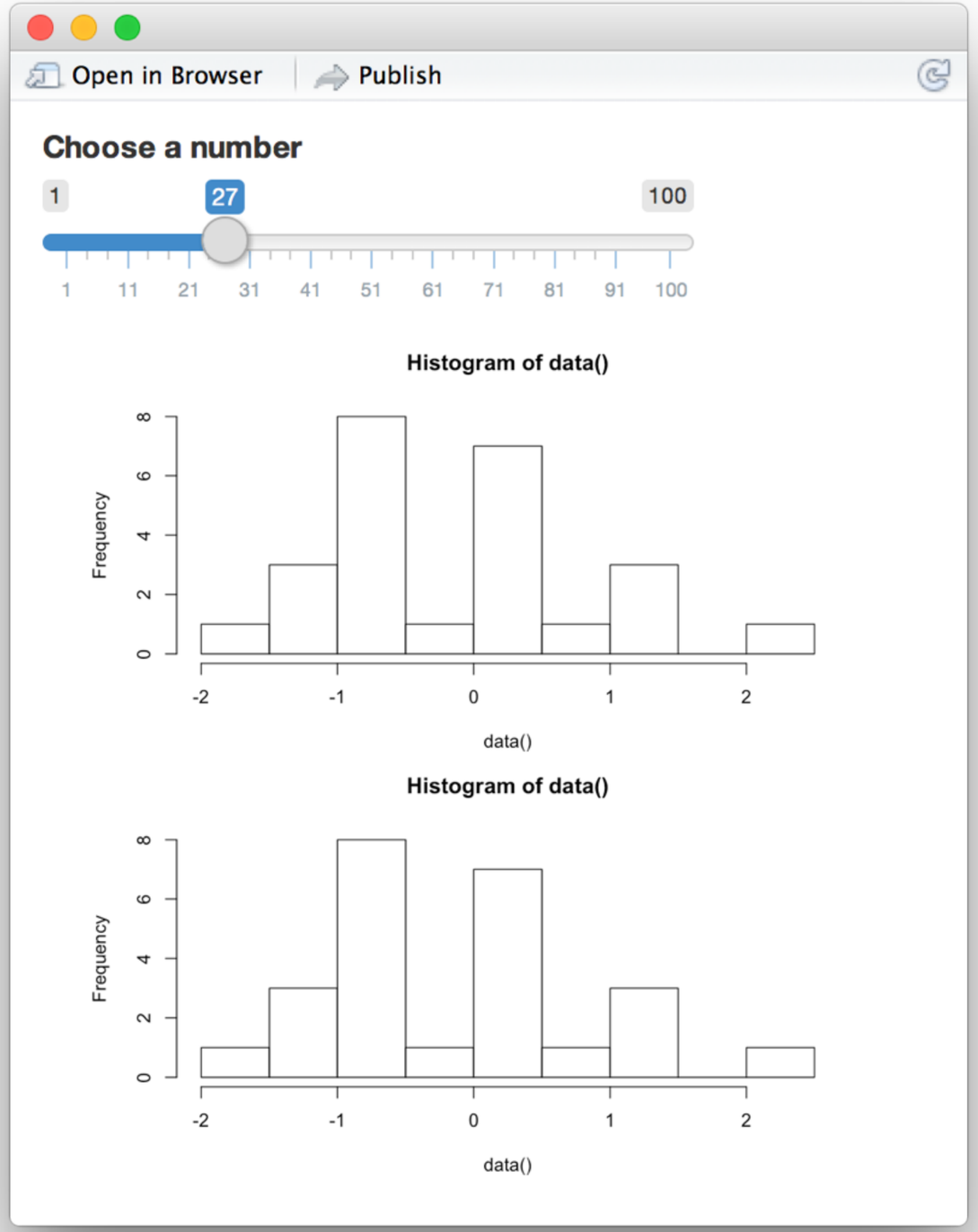
```
library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist"),
  plotOutput("hist2")
)
server <- function(input, output) {
  data <- reactive(rnorm(input$num))
  output$hist <- renderPlot({
    hist(data())
  })
  output$hist2 <- renderPlot({

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

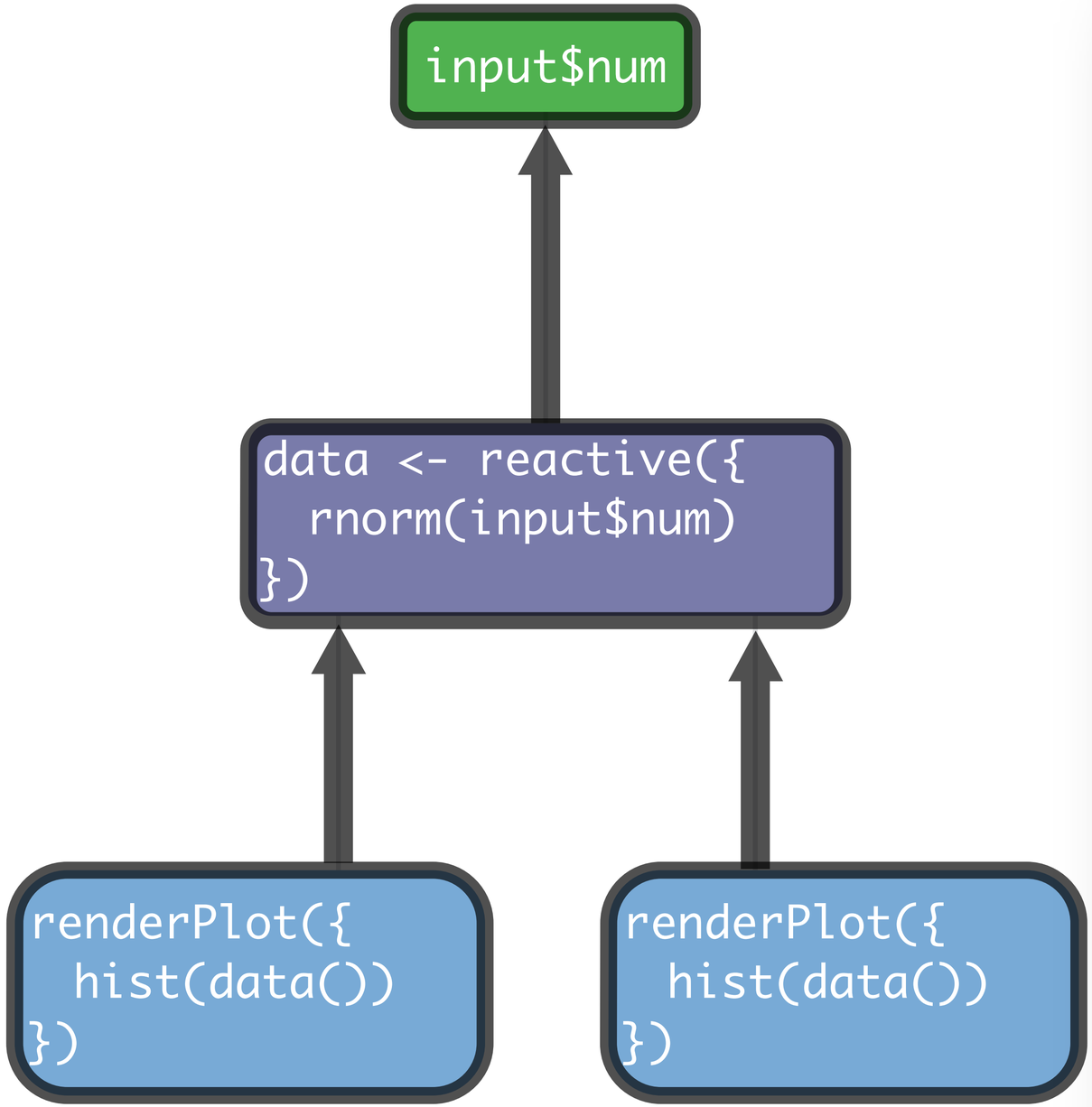


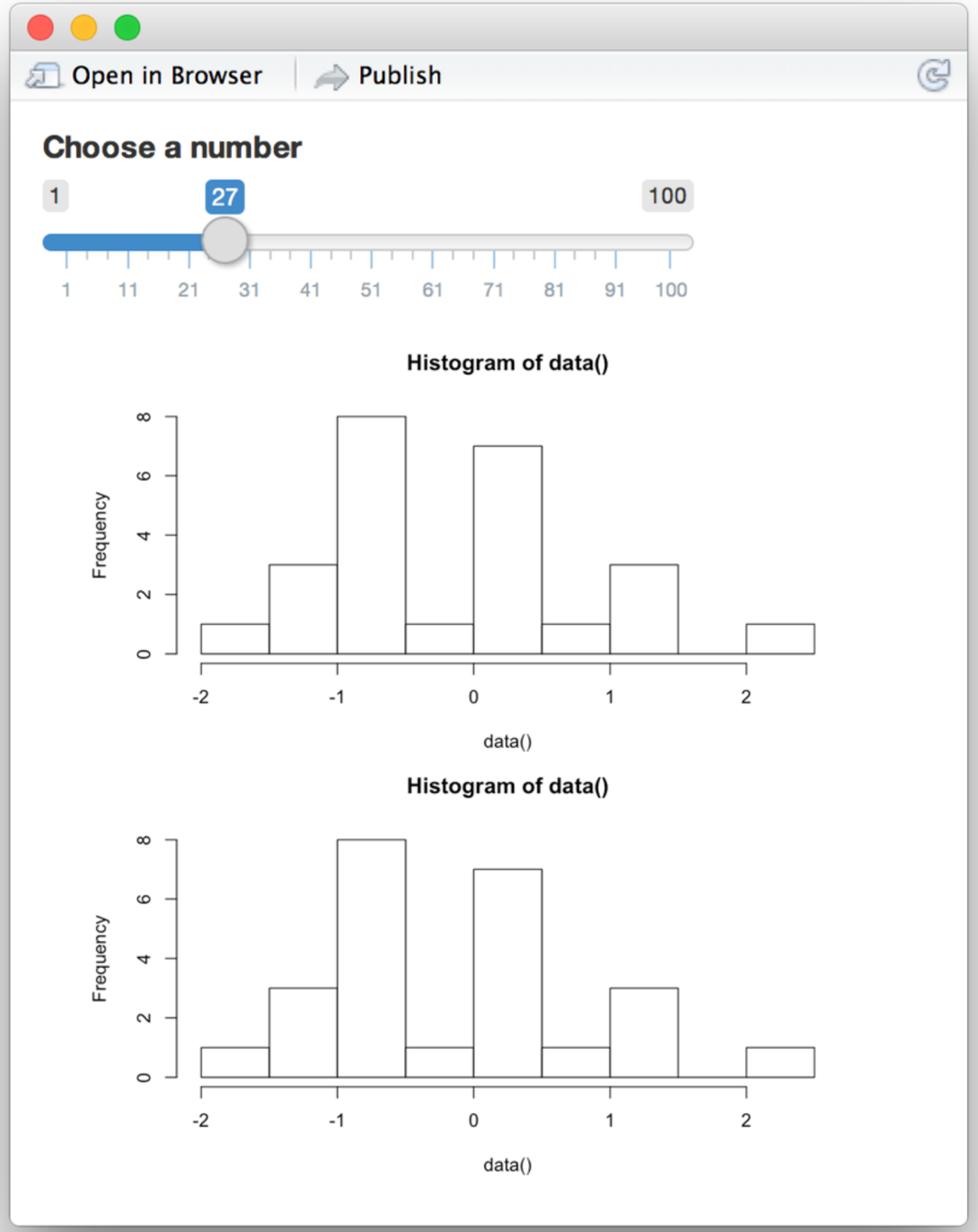
```
library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist"),
  plotOutput("hist2")
)
server <- function(input, output) {
  data <- reactive(rnorm(input$num))
  output$hist <- renderPlot({
    hist(data())
  })
  output$hist2 <- renderPlot({
    hist(data())
  })
}
shinyApp(ui = ui, server = server)
```









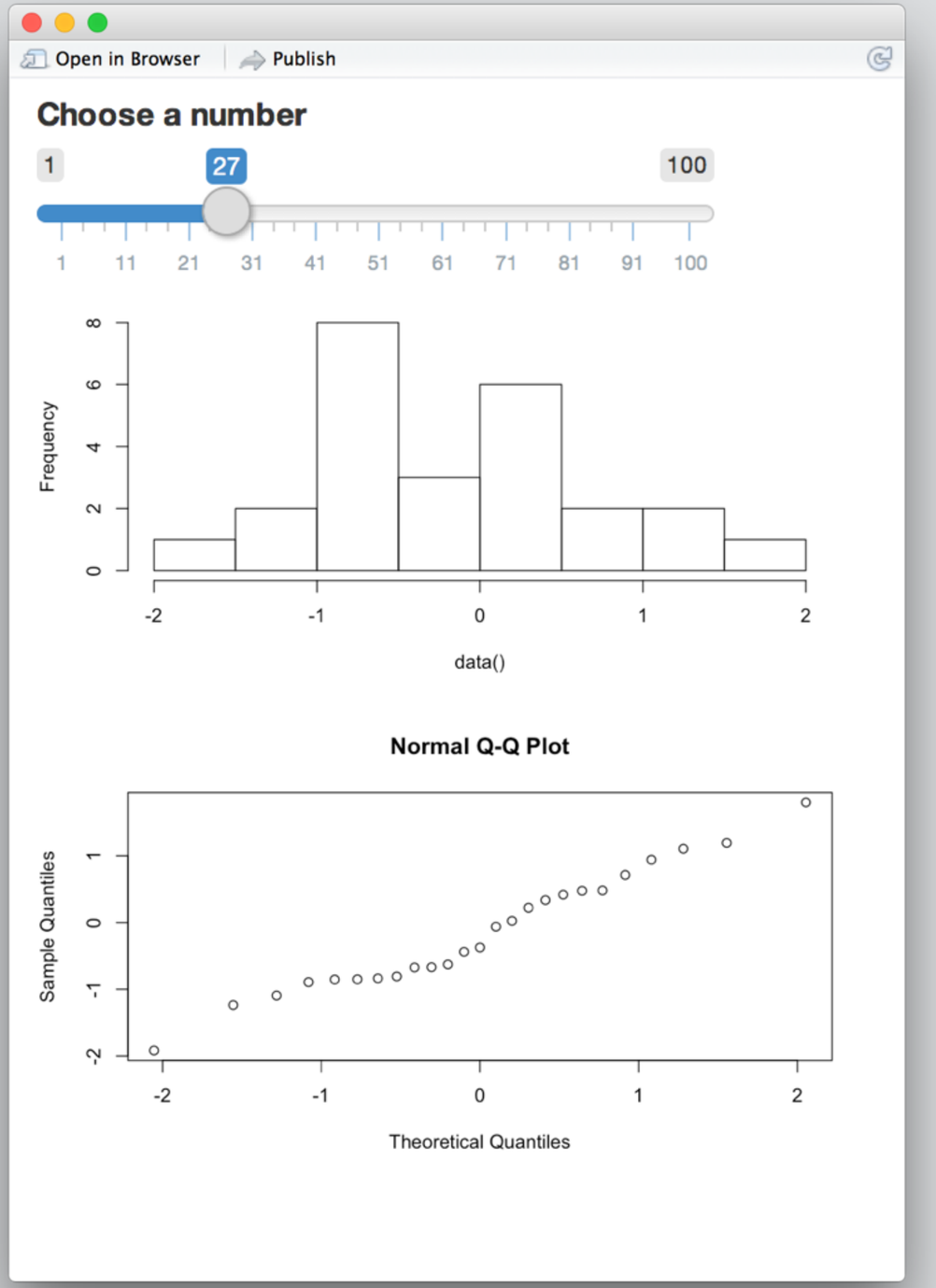


A reactive expression is special in two ways

```
data()
```

1 You call a reactive expression like a function

2 Reactive expressions **cache** their values
(if the expression knows that it is up-to-date, it will return its most recent value, which is saved in memory)



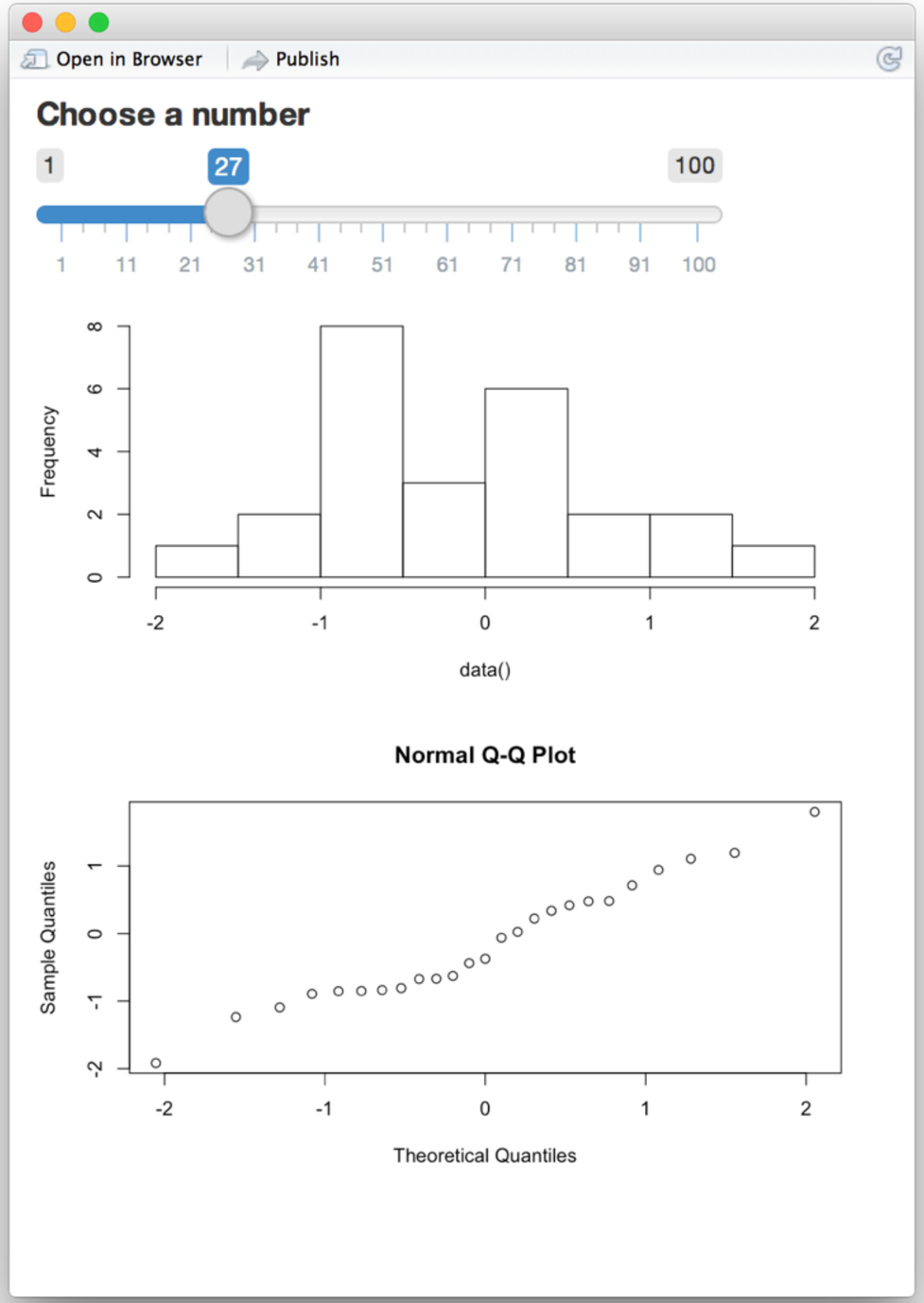
Your turn

Rewrite your app to use a reactive expression.

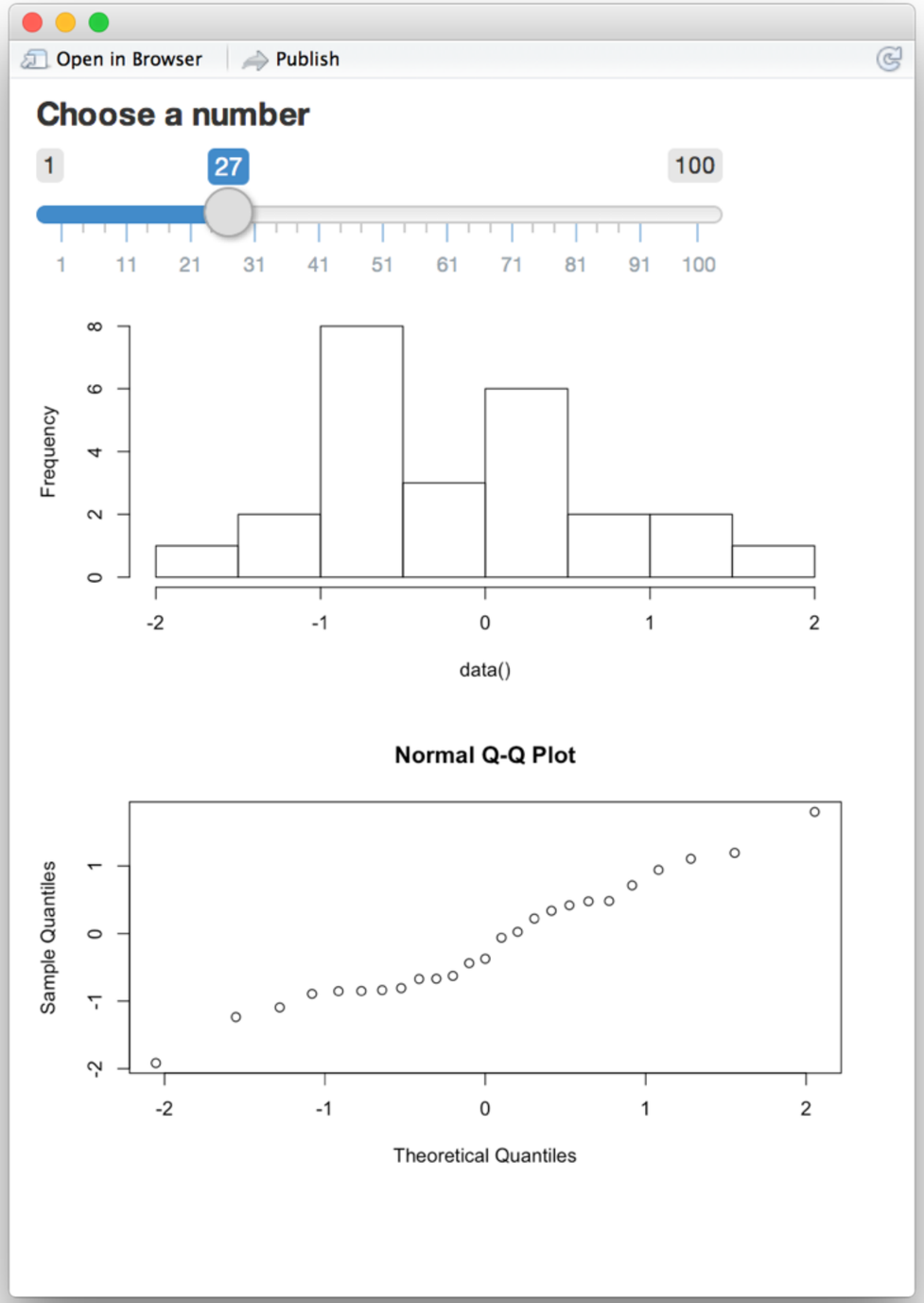
Make sure the histogram and qqplot display the same data.

05:00

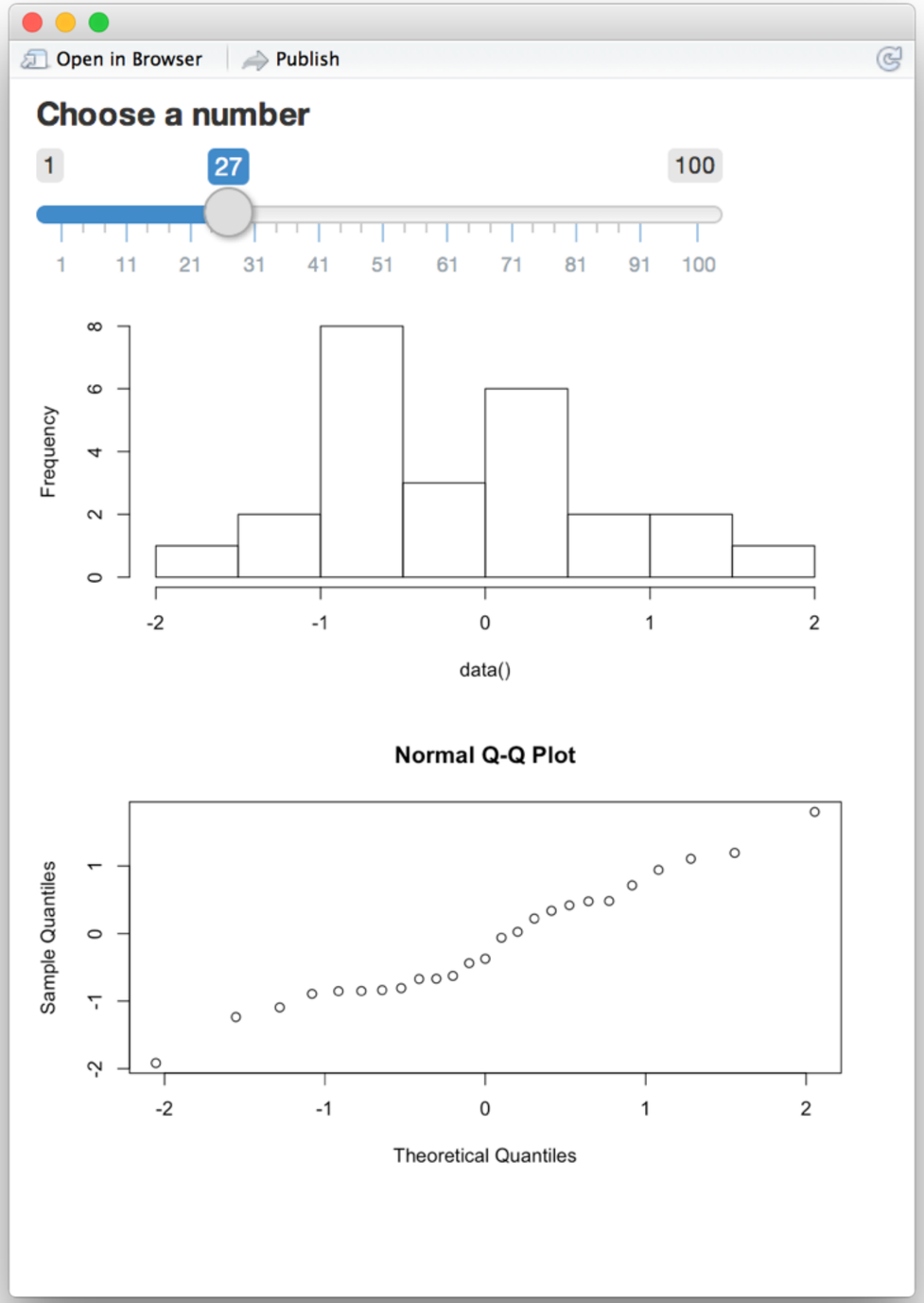
```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist"),
  plotOutput("qq")
)

server <- function(input, output) {

  output$hist <- renderPlot({hist( )})
  output$qq <- renderPlot({qqnorm( )})
}

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



```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist"),
  plotOutput("qq")
)

server <- function(input, output) {
  data <- reactive(rnorm(input$num))
  output$hist <- renderPlot({hist( )})
  output$qq <- renderPlot({qqnorm( )})
}

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



```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist"),
  plotOutput("qq")
)

server <- function(input, output) {
  data <- reactive(rnorm(input$num))
  output$hist <- renderPlot({hist(data())})
  output$qq <- renderPlot({qqnorm(data())})
}

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



isolate()

isolate()

Returns the result as a non-reactive value

```
isolate({ rnorm(input$num) })
```

a shiny function
(can handle reactive values)

code block
to run

code will NOT rerun when a
reactive value changes

observeEvent()

observeEvent()

Triggers code to run on server

```
observeEvent(input$num, { print(input$num) })
```

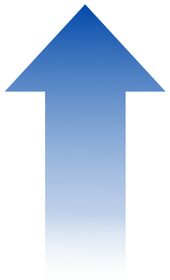
a shiny function
(can handle reactive values)

a reactive value to monitor

code block to run whenever the value changes

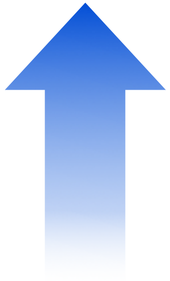
observeEvent()

```
observeEvent(input$num, { print(input$num) })
```



2 When will it run it?

Each time this value changes

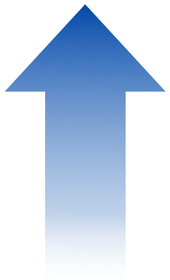


1 What code will the server run?

All of the code in the code block

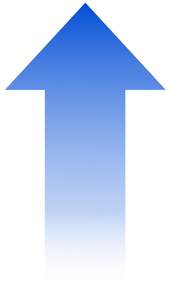
observeEvent()

```
observeEvent(input$num2, { print(input$num) })
```



2 When will it run it?

Each time this value changes

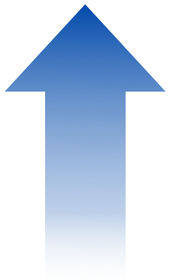


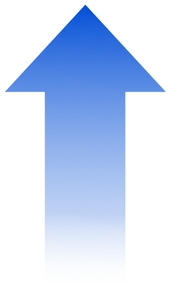
1 What code will the server run?

All of the code in the code block

observeEvent()

```
observeEvent(input$num2, { print("Hello") })
```





2 When will it run it?

Each time this value changes

1 What code will the server run?

All of the code in the code block

eventReactive()

eventReactive()

A reactive expression that remains valid until a specific value changes

```
data <- eventReactive(input$go,  { rnorm(input$num) })
```

A function to call

a shiny function
(can handle reactive values)

A reactive value to respond to
(expression invalidates ONLY when this value changes)

code block to run when data() is called

Recap

Recap: Reactive toolkit

`render*()`	Build reactive output to display in UI
`reactive()`	Build reactive expression to use in code
`isolate()`	Build non-reactive result to use in code
`observeEvent()`	Trigger code on server
`eventReactive()`	Build expression that only reacts to set value(s)

Layout

2

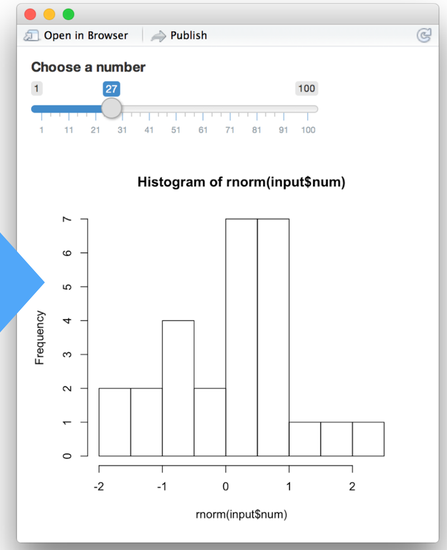
Add elements to your app as arguments to `fluidPage()`

```
fluidPage(
  # input controls,
  # output controls,
  # layout objects,
  # custom HTML/CSS/JavaScript
)
```

Shiny's UI functions return HTML

```
fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    val = 25, min = 1, max = 100),
  plotOutput("hist")
)
```

```
<div class="container-fluid">
  <div class="form-group shiny-input-container">
    <label class="control-label"
      for="num">Choose a number</label>
    <input class="js-range-slider" id="num"
      data-min="1" data-max="100"
      data-from="25"/>
  </div>
  <div id="hist" class="shiny-plot-output"
    style="width: 100% ; height: 400px"></div>
</div>
```



Layout functions

Add HTML that divides the UI into a grid

```
fluidRow()
```

```
<div class="row"></div>
```

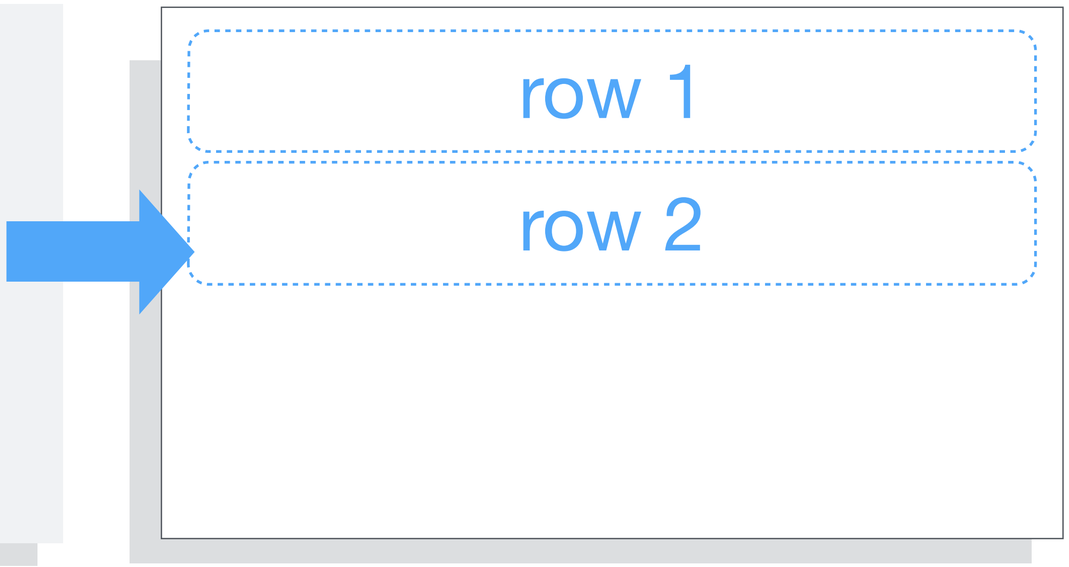
```
column(width = 2)
```

```
<div class="col-sm-2"></div>
```

fluidRow()

`fluidRow()` adds rows to the grid. Each new row goes below the previous rows.

```
ui <- fluidPage(
  fluidRow(),
  fluidRow()
)
```

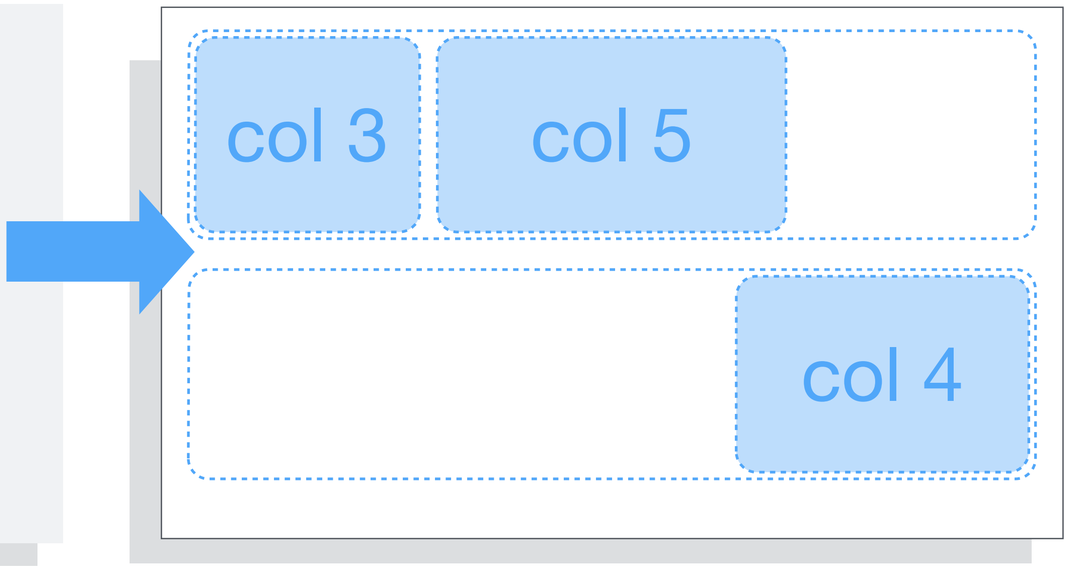


column()

`column()` adds columns within a row. Each new column goes to the left of the previous column.

Specify the **width** and **offset** of each column out of 12

```
ui <- fluidPage(
  fluidRow(
    column(3),
    column(5)),
  fluidRow(
    column(4, offset = 8)
)
```



To place an element in the grid, call it as an argument of a layout function

```
fluidRow("In the row")
```

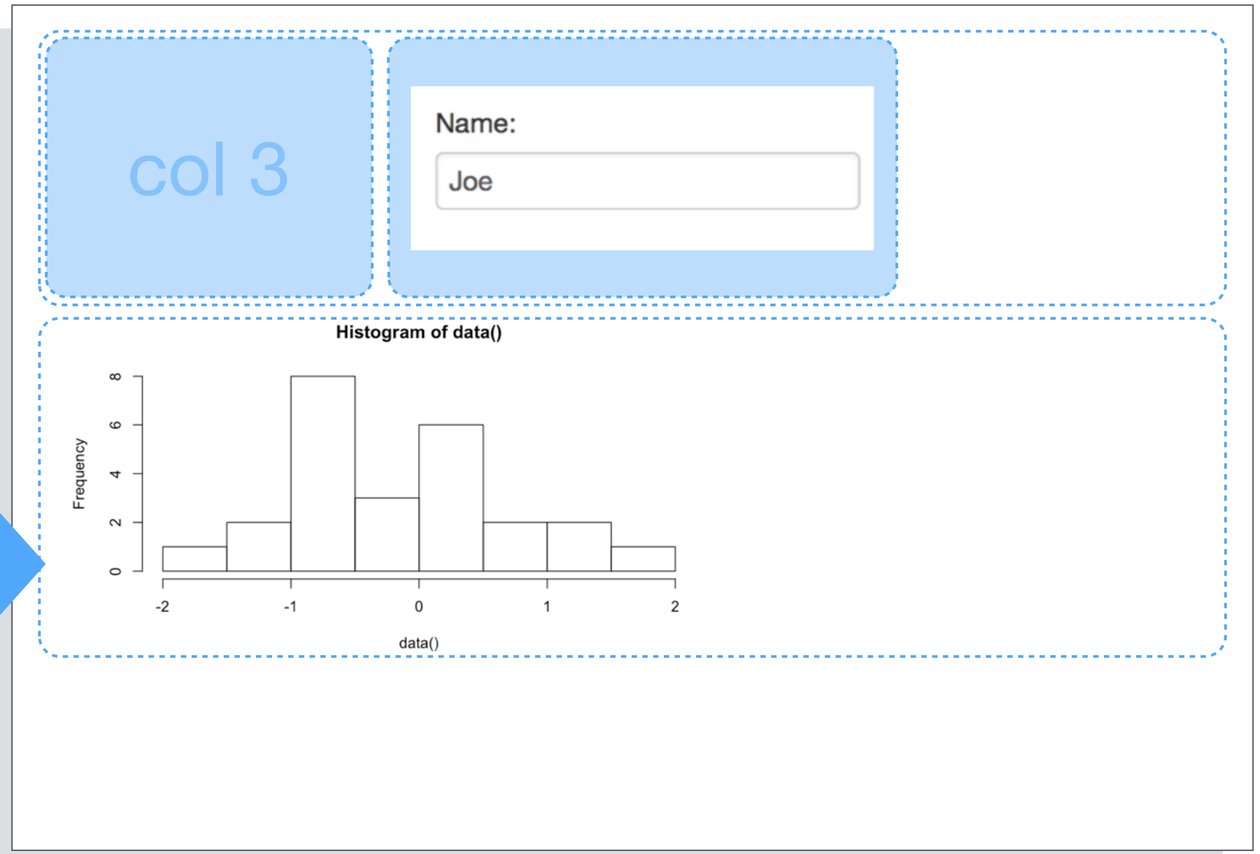
```
<div class="row">In the row</div>
```

```
column(plotOutput("hist"))
```

```
<div class="col-sm-2">
  <div id="hist" class="shiny-plot-output"
  style="width: 100% ; height: 400px"></div>
</div>
```

To place an element in the grid, call it as an argument of a layout function

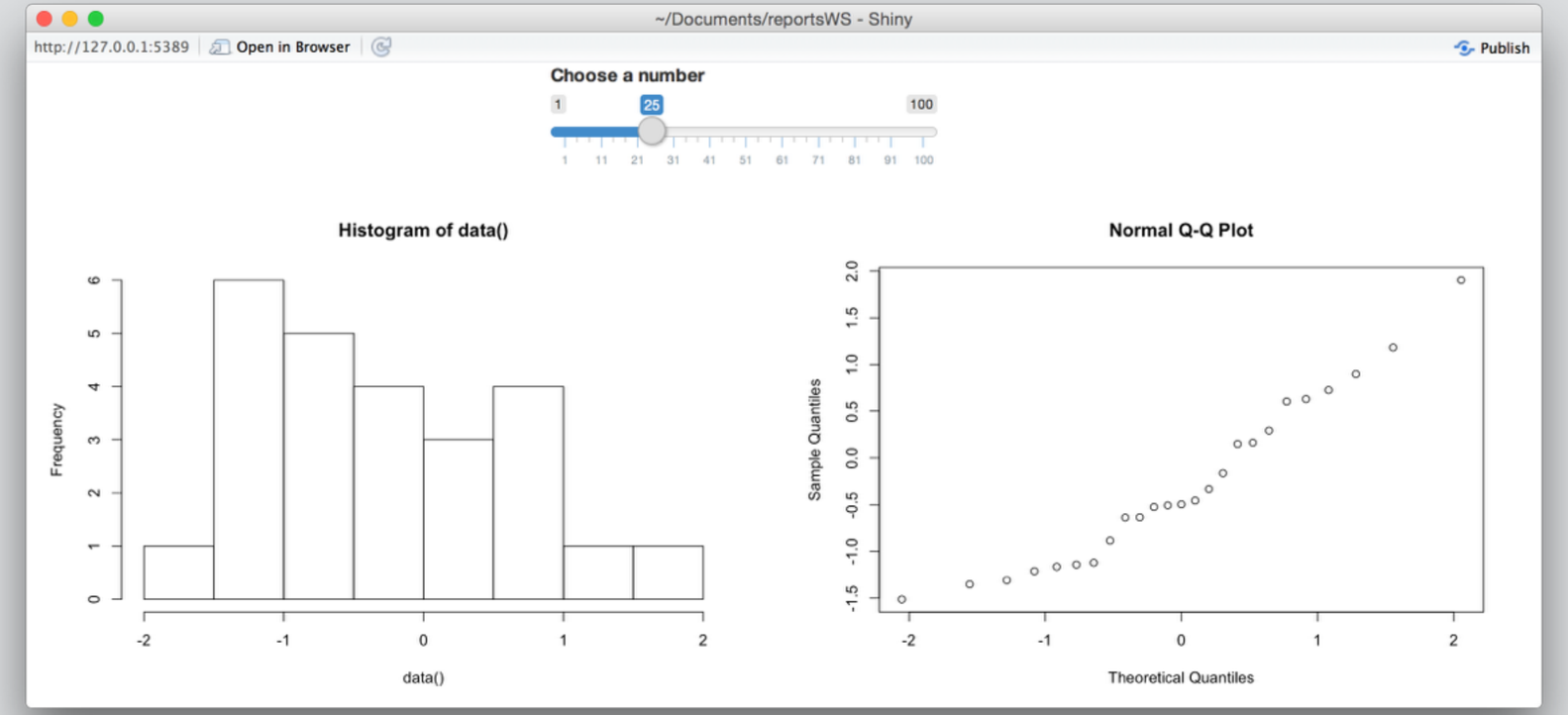
```
fluidPage(
  fluidRow(
    column(3),
    column(5,
      textInput("n", "Name:")
    )
  ),
  plotOutput("hist")
)
```



Your turn

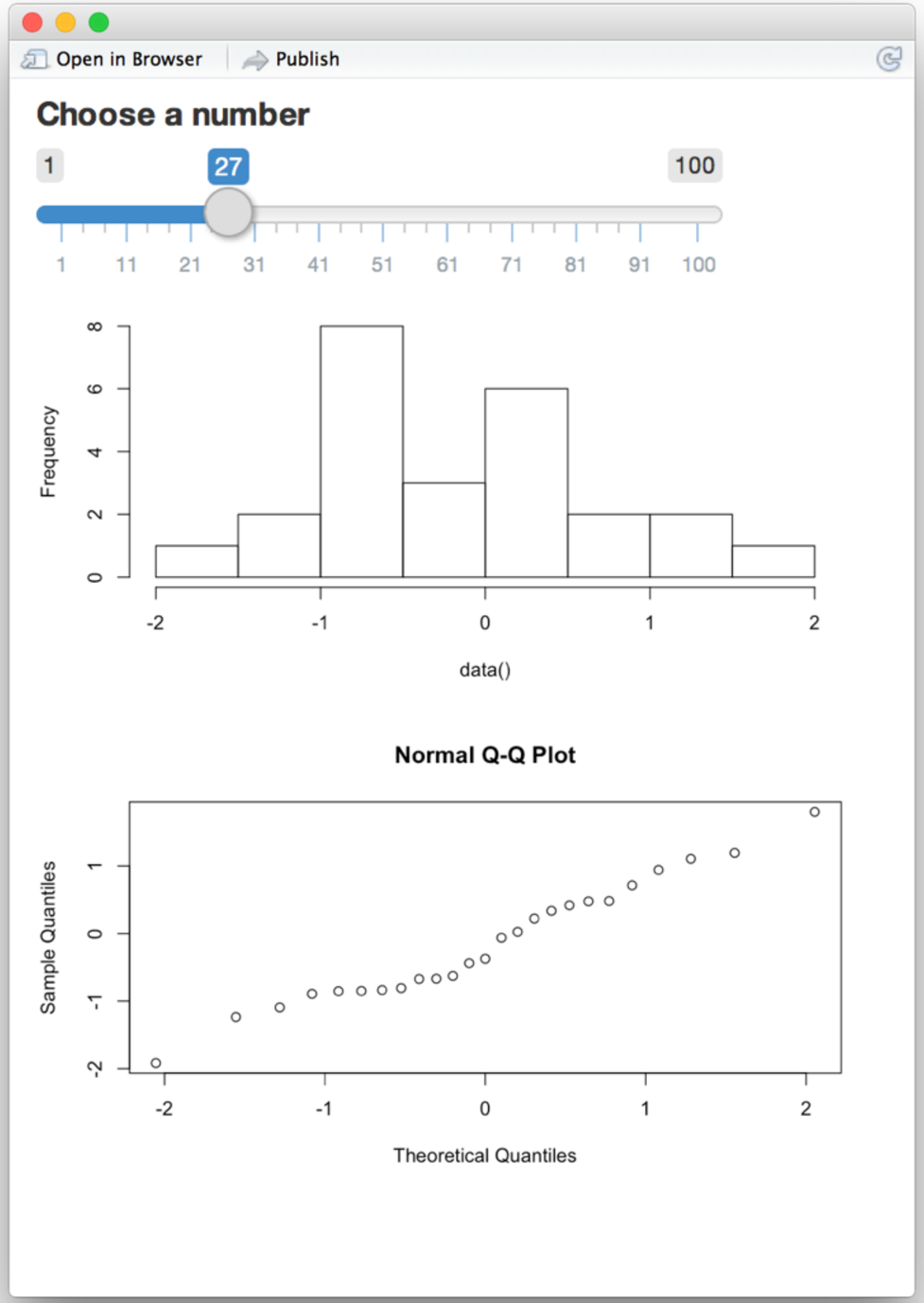
Return to the last exercise.

Use `fluidRow()` and `column()` to lay your app out like this.

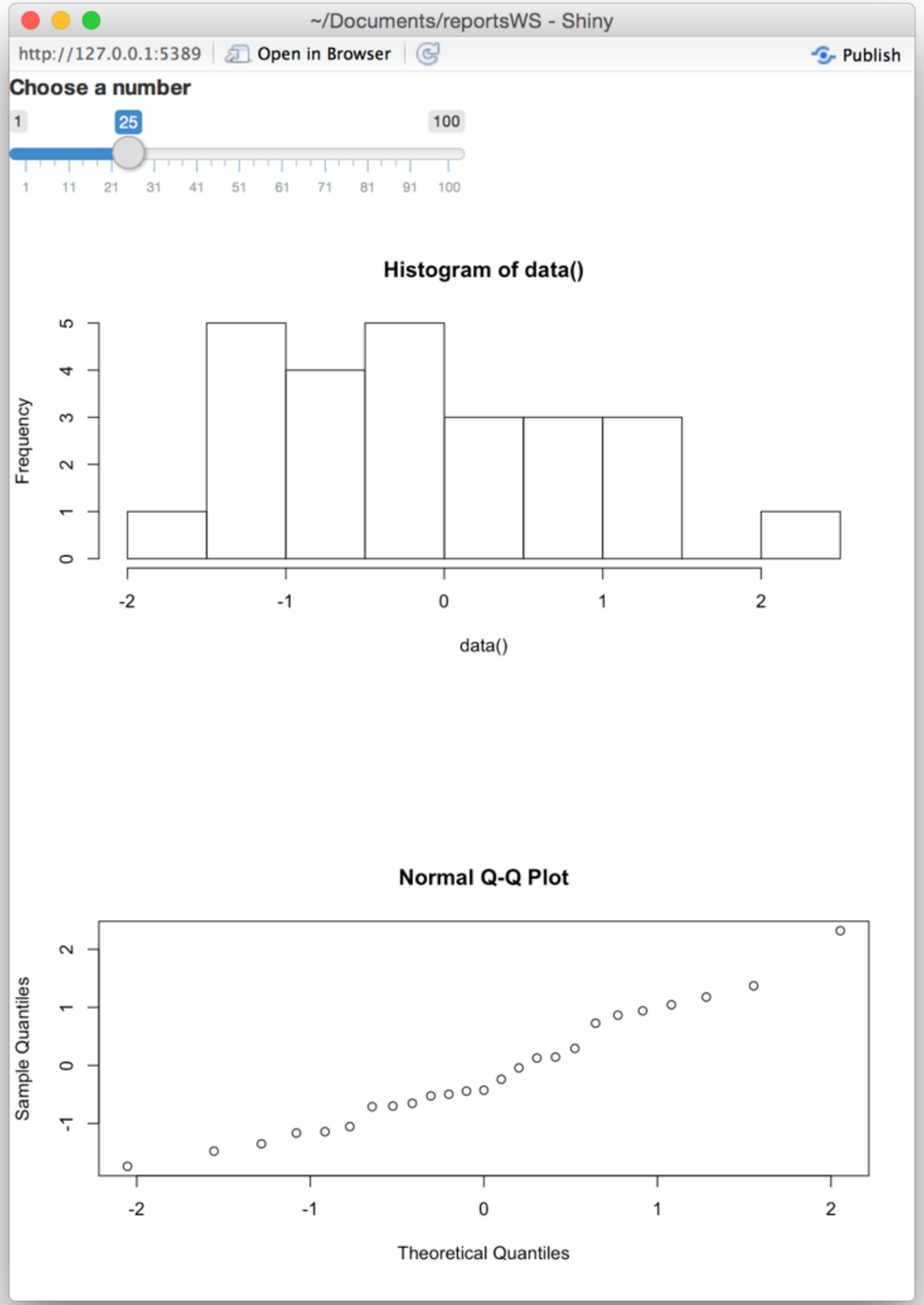


Hint: You do not need to change anything in server

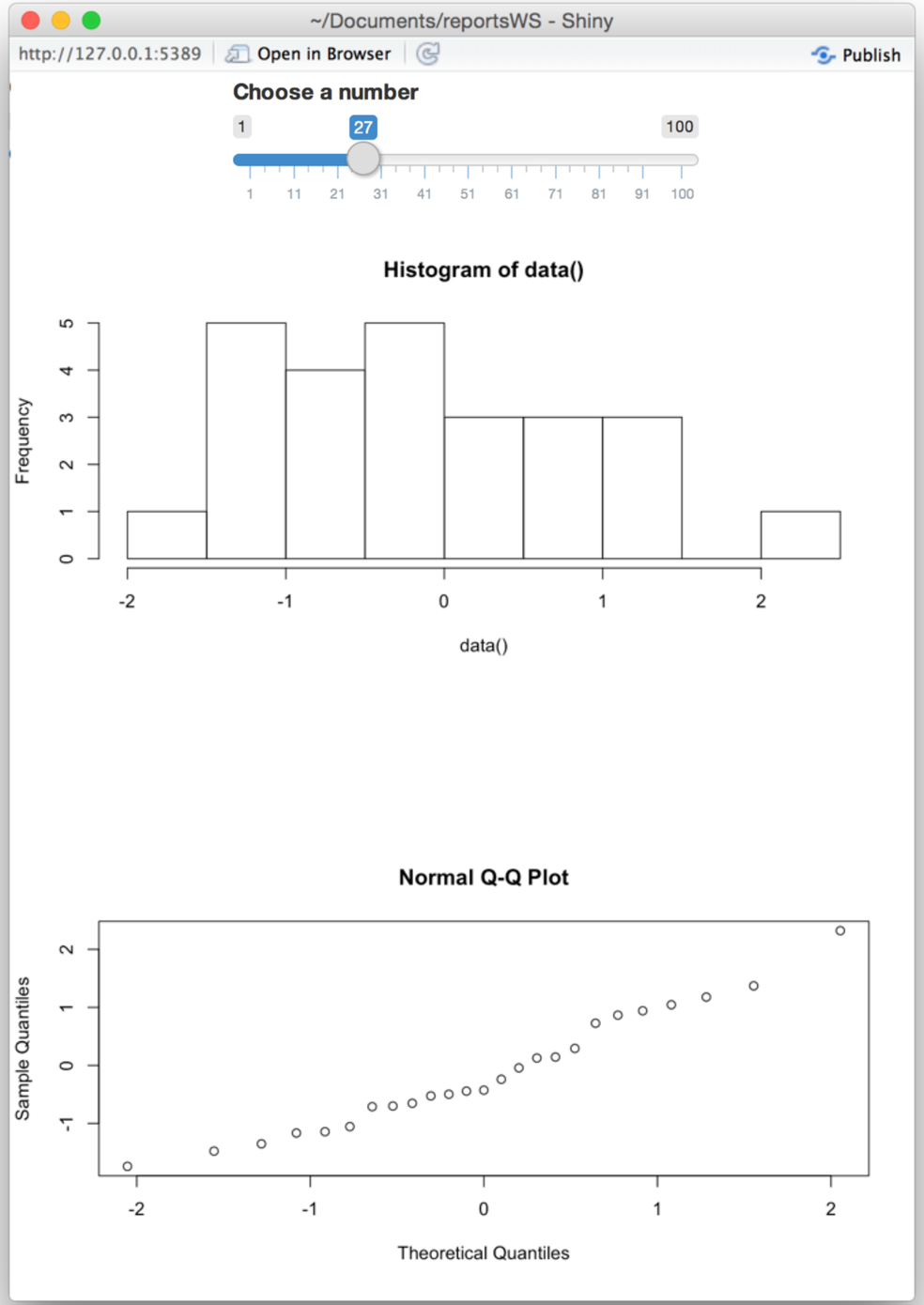
```
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    val = 25, min = 1, max = 100),
  plotOutput("hist"),
  plotOutput("qq")
)
```



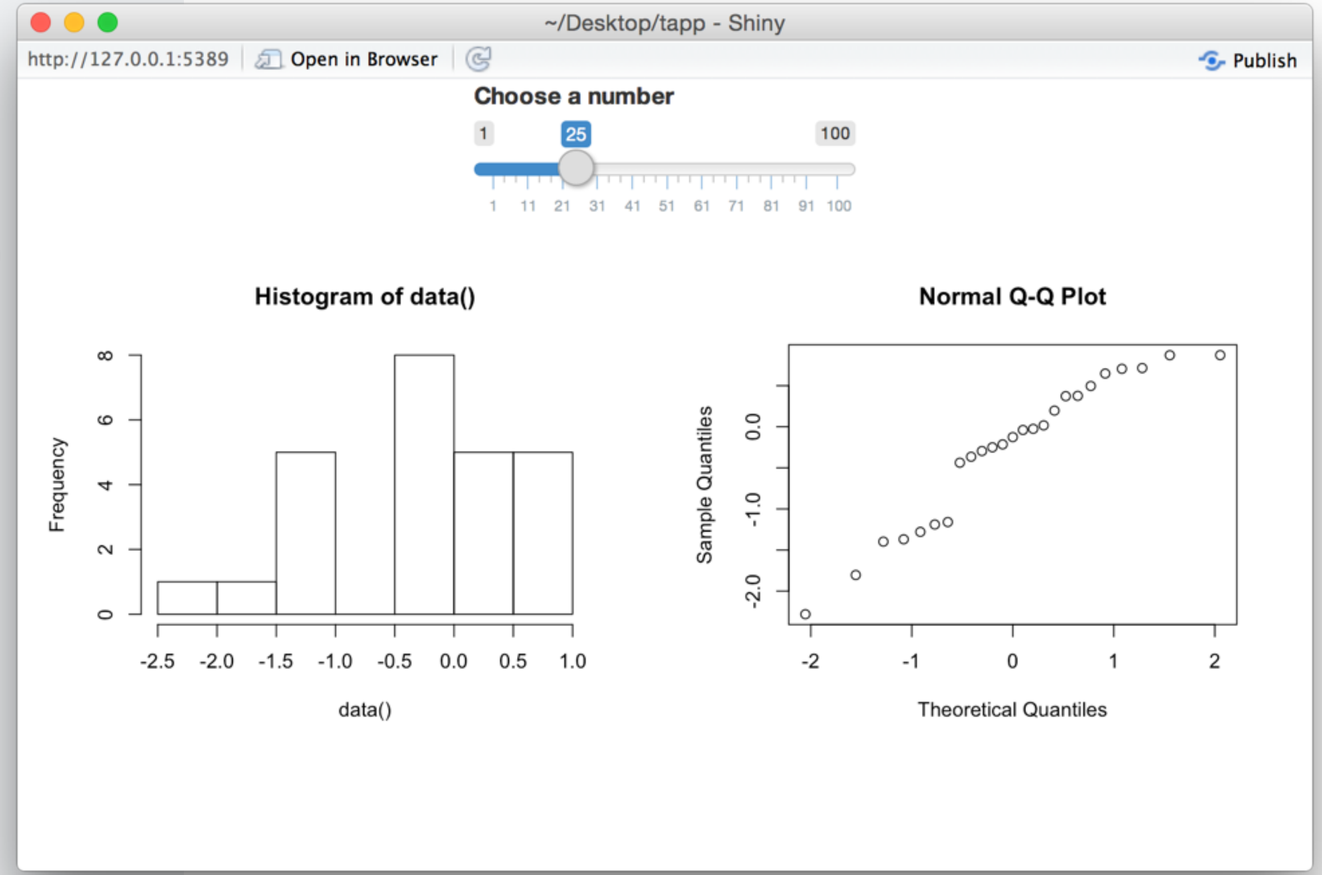
```
ui <- fluidPage(
  fluidRow(
    sliderInput(inputId = "num",
      label = "Choose a number",
      val = 25, min = 1, max = 100)
  ),
  fluidRow(
    plotOutput("hist"),
    plotOutput("qq")
  )
)
```



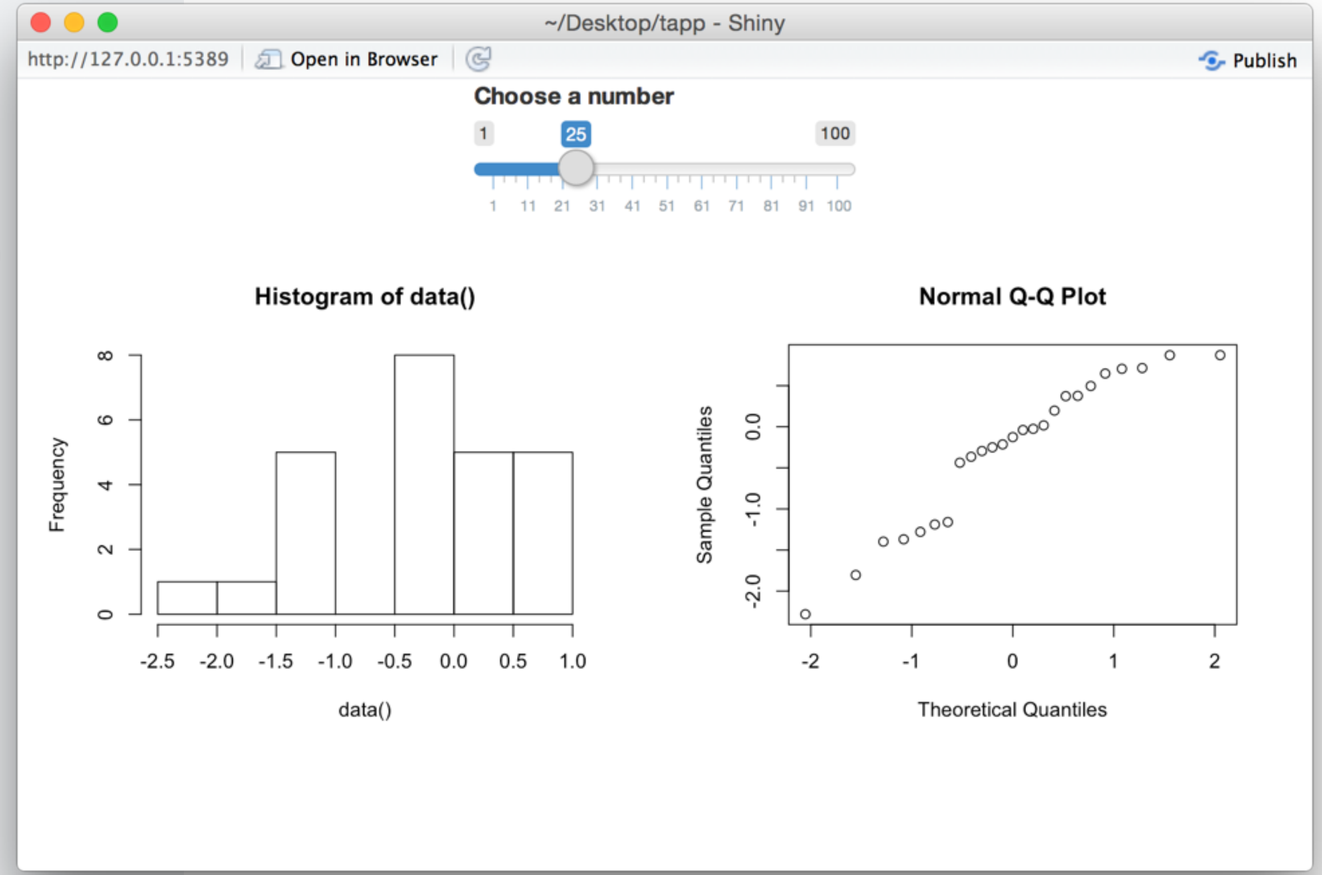
```
ui <- fluidPage(
  fluidRow(
    column(width = 4, offset = 4,
      sliderInput(inputId = "num",
        label = "Choose a number",
        val = 25, min = 1, max = 100)
    )
  ),
  fluidRow(
    plotOutput("hist"),
    plotOutput("qq")
  )
)
```



```
ui <- fluidPage(
  fluidRow(
    column(width = 4, offset = 4,
      sliderInput(inputId = "num",
        label = "Choose a number",
        val = 25, min = 1, max = 100)
    )
  ),
  fluidRow(
    column(6, plotOutput("hist")),
    column(6, plotOutput("qq"))
  )
)
```



```
ui <- fluidPage(
  fluidRow(
    column(width = 4, offset = 4,
      sliderInput(inputId = "num",
        label = "Choose a number",
        val = 25, min = 1, max = 100)
    )
  ),
  fluidRow(
    column(6, plotOutput("hist")),
    column(6, plotOutput("qq"))
  )
)
```

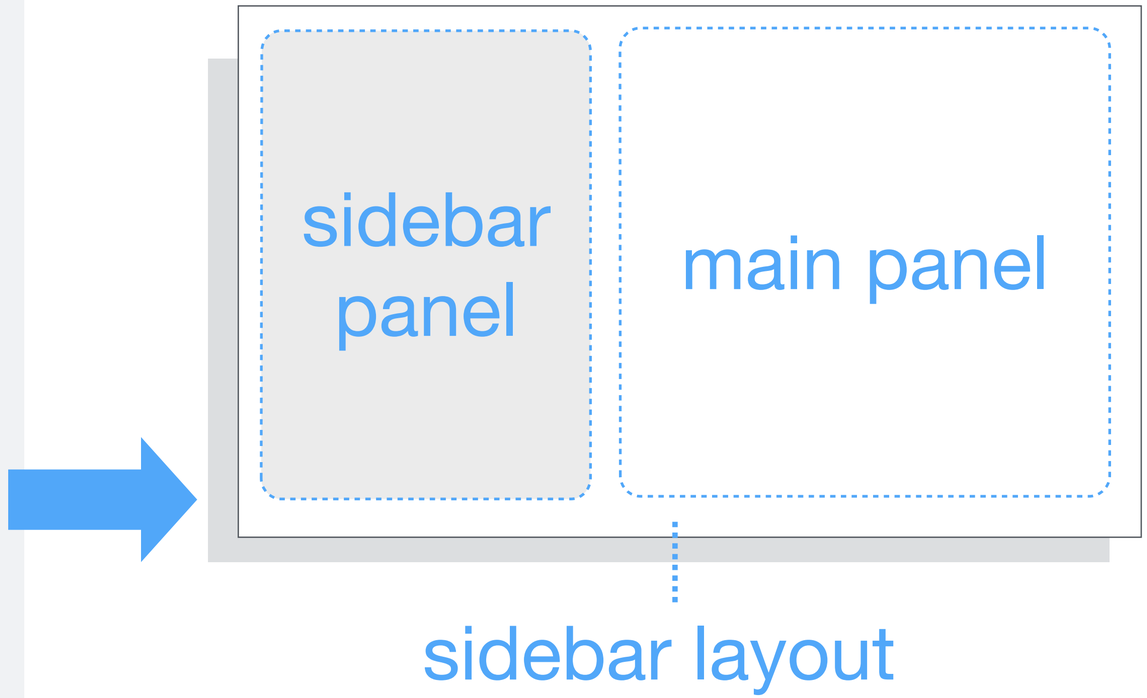


Prepackaged Layouts

pageWithSidebar()

The most simple way to layout a Shiny app. Use with titlePanel(), sidebarPanel() and mainPanel()

```
ui <- pageWithSidebar(
  titlePanel(),
  sidebarPanel(),
  mainPanel()
)
```

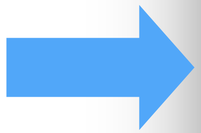


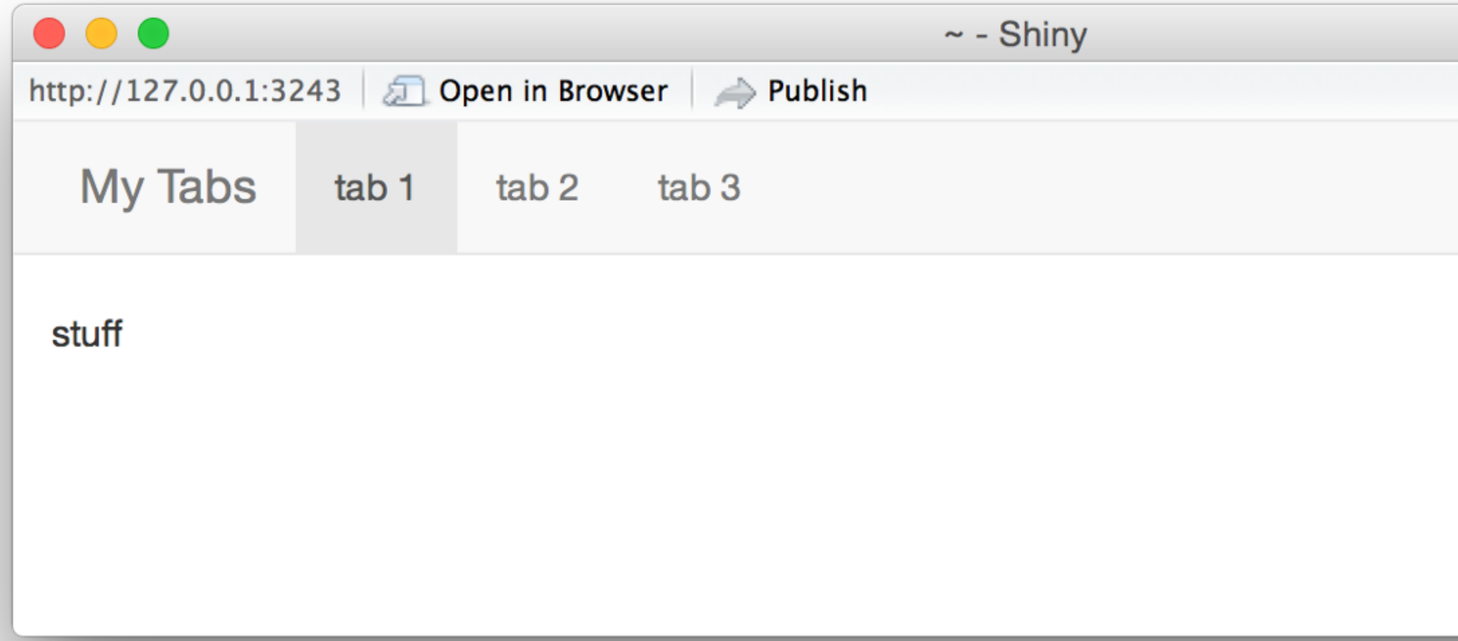
navbarPage()

`navbarPage()` combines tabs into a single *page*.

navbarPage() replaces fluidPage().

```
ui <- navbarPage("My Tabs",
   tabPanel("tab 1", "stuff"),
   tabPanel("tab 2", "stuff"),
   tabPanel("tab 3", "stuff")
)
```

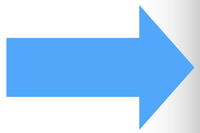


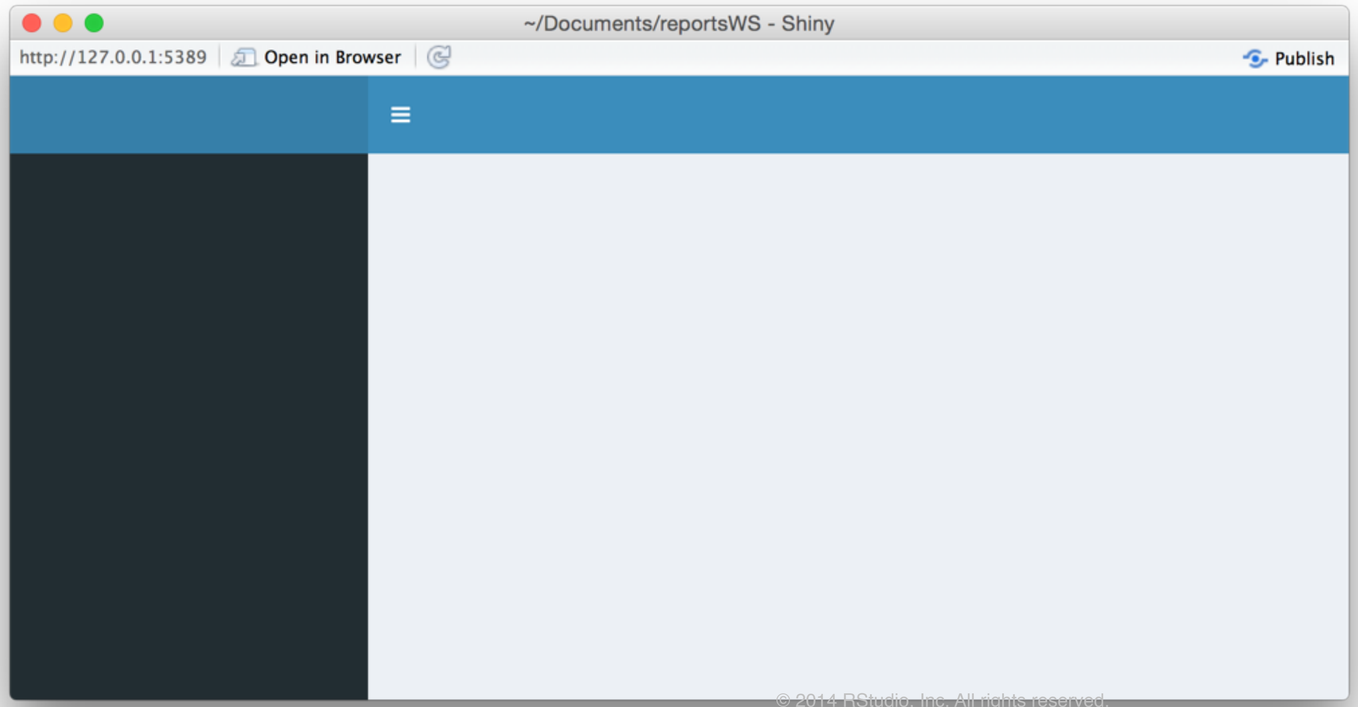


dashboardPage()

`dashboardPage()` comes in the shinydashboard package

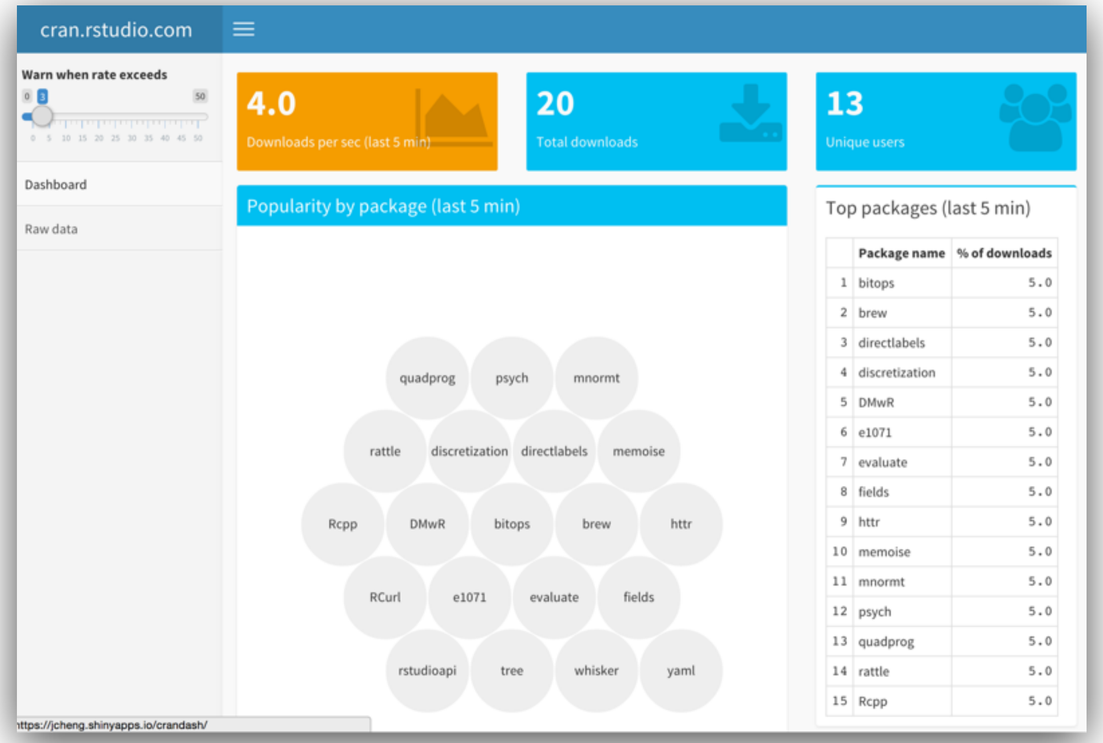
```
library(shinydashboard)
ui <- dashboardPage(
  dashboardHeader(),
  dashboardSidebar(),
  dashboardBody()
)
```





shinydashboard

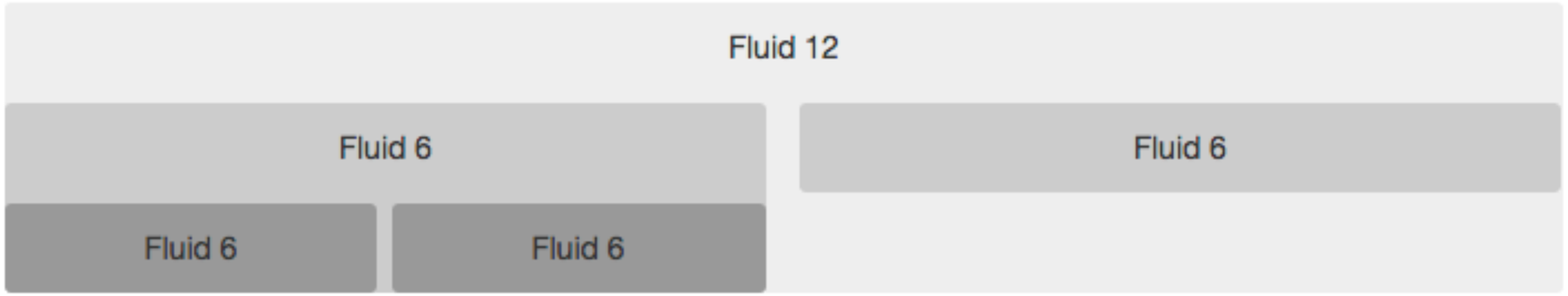
http://rstudio.github.io/shinydashboard/



A package of layout functions for building administrative dashboards with Shiny

The Shiny Layout Guide

http://shiny.rstudio.com/articles/layout-guide.html



You can build sophisticated, customized layouts with Shiny's grid system.

Custom HTML/CSS/JS

Customize your UI with HTML

http://shiny.rstudio.com/articles/html-tags.html

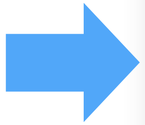
You can add any HTML element to your app with one of Shiny's `tags` function.

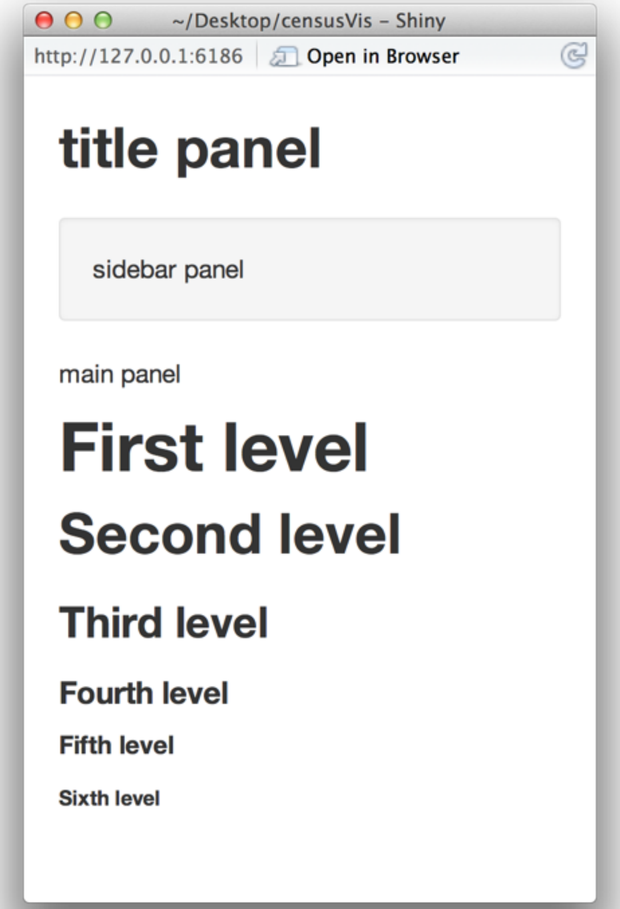
```
names(tags)
##   [1] "a"          "abbr"       "address"    "area"       "article"
##   [6] "aside"      "audio"      "b"          "base"       "bdi"
##  [11] "bdo"        "blockquote" "body"       "br"         "button"
##  [16] "canvas"     "caption"    "cite"       "code"       "col"
##  [21] "colgroup"   "command"    "data"       "datalist"   "dd"
##  [26] "del"        "details"    "dfn"        "div"        "dl"
##  [31] "dt"         "em"         "embed"      "eventsource" "fieldset"
```

h1() - h6()

Headers

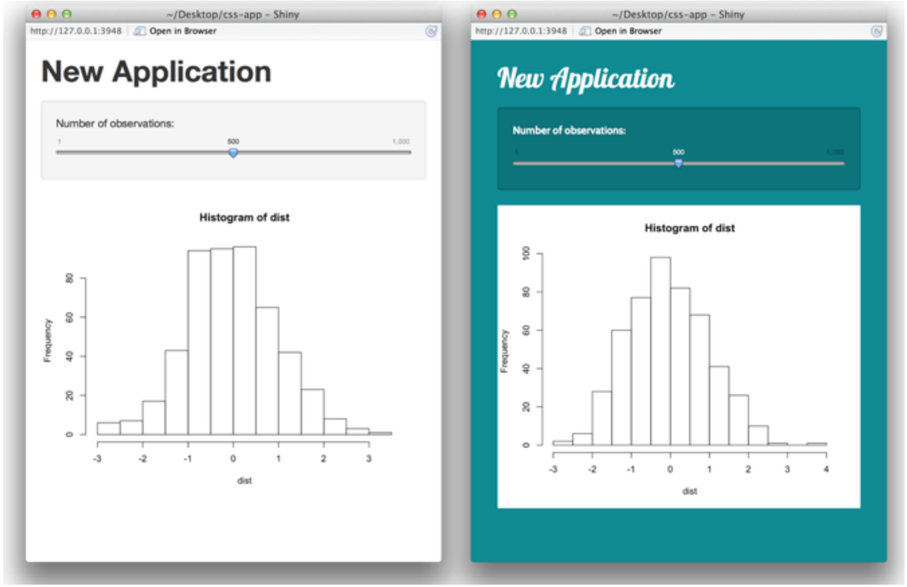
```
fluidPage(
  titlePanel("title panel"),
  sidebarLayout(
    sidebarPanel("sidebar panel"),
    mainPanel("main panel",
      h1("First level"),
      h2("Second level"),
      h3("Third level"),
      h4("Fourth level"),
      h5("Fifth level"),
      h6("Sixth level"))
  )
)
```





Customize your apps with HTML, CSS, and Javascript

http://shiny.rstudio.com/articles/css.html



You can pair any app with whatever web technologies you wish. The above guide explains how to style your app with CSS.

HTML UI

shiny.rstudio.com/articles/html-ui.html

Build your entire UI from HTML

```
<html>

<head>
  <script src="shared/jquery.js" type="text/javascript"></script>
  <script src="shared/shiny.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="shared/shiny.css"/>
</head>

<body>
  <h1>HTML UI</h1>

  <p>
    <label>Distribution type:</label><br />
```

Sharing Apps

shinyapps.io

Shiny Server

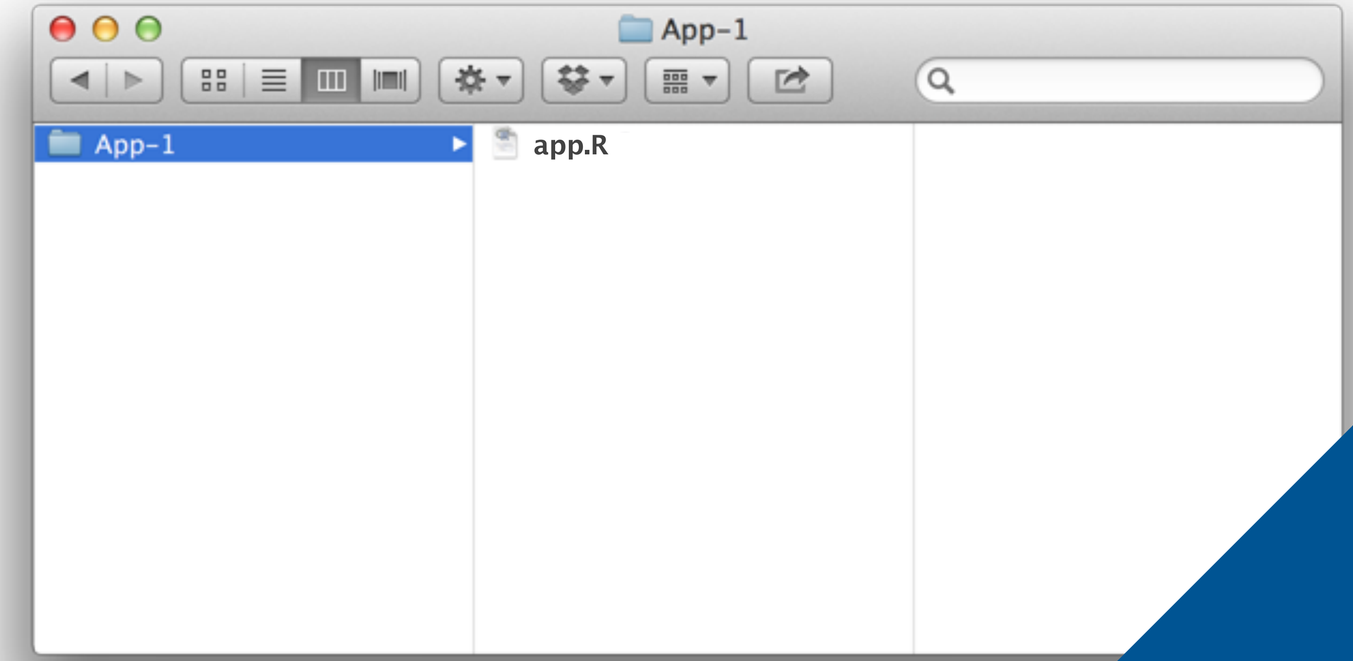
Shiny Server Pro

App Directories

Structure of a Shiny web app

One directory with every file the app needs:

- app.R *(your script which ends with a call to shinyApp())*
- datasets, images, css, helper scripts, etc.



You must use this exact name (**app.R**)

runApp

You can launch the app with `runApp` (your computer will build a local web site that hosts the app).

```
runApp("~/Documents/App-1")
```

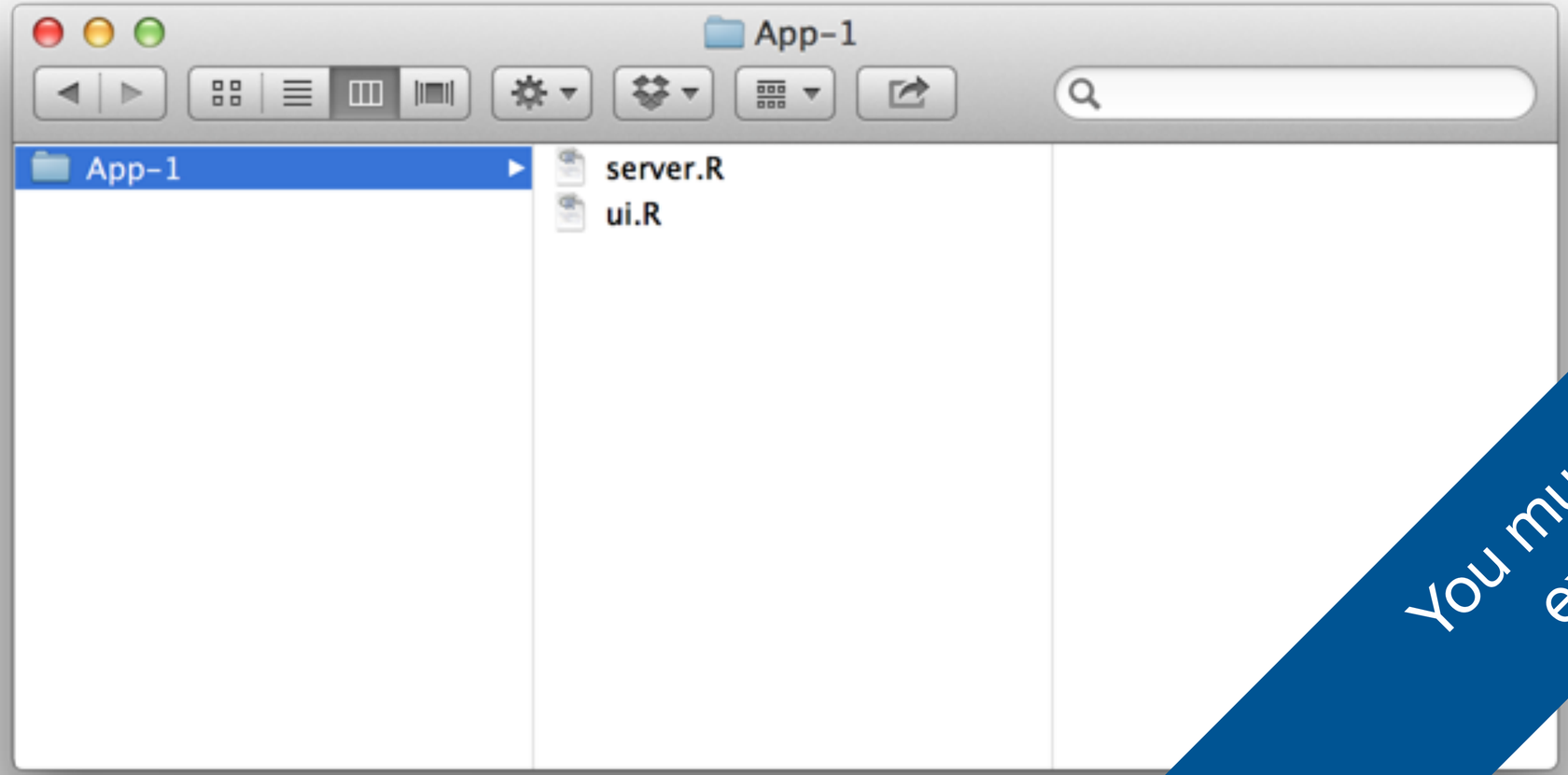
File path to app directory.

R will append the file path to the working directory, if path does not begin at the home directory

Two file apps

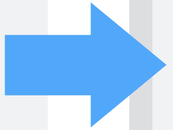
One directory with two files:

- server.R
- ui.R



You must use these exact names

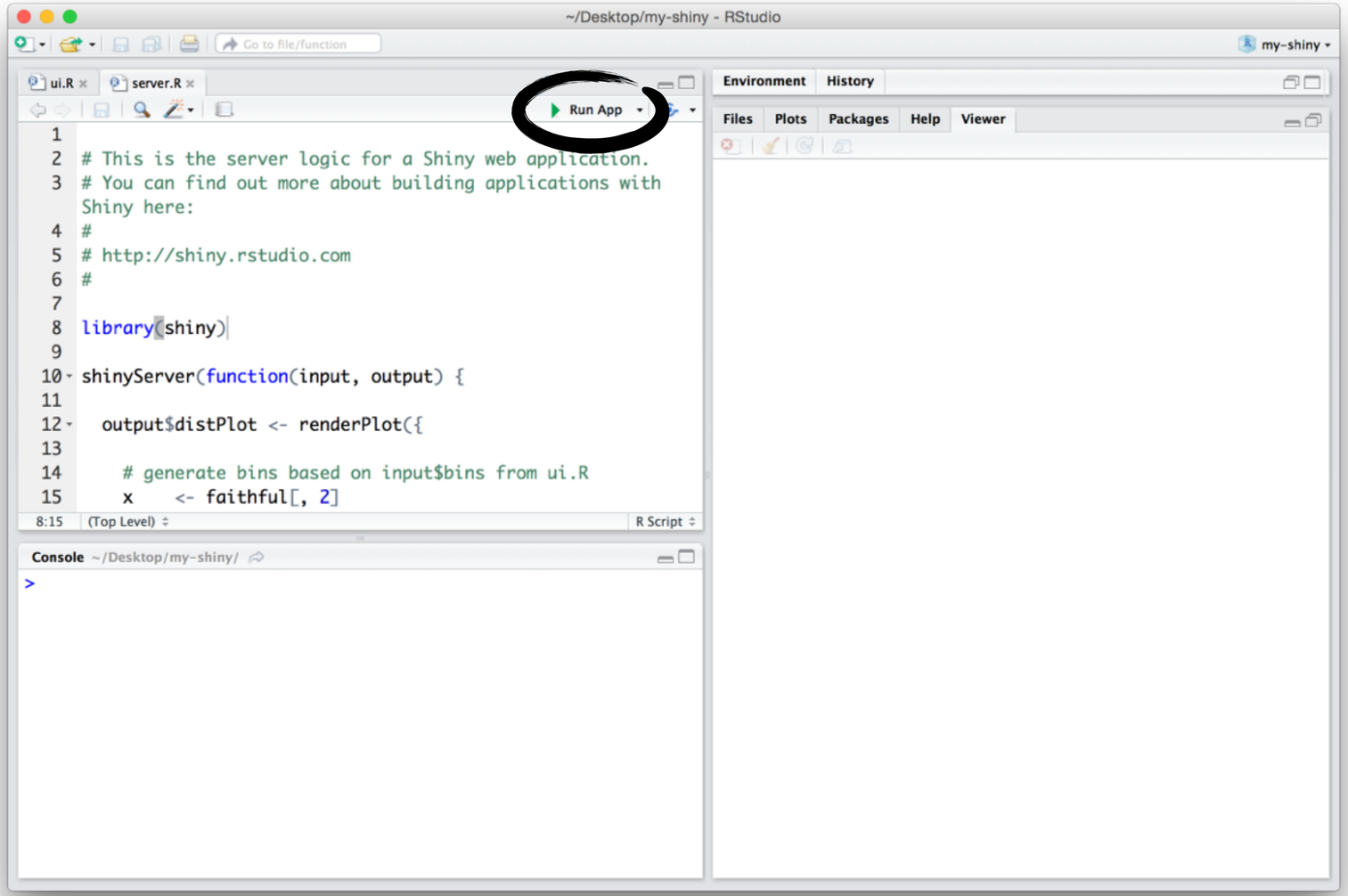
```
library(shiny)
server <- function(input, output) {
  output$distPlot <- renderPlot({
    x    <- faithful[, 2]
    bins <- seq(min(x), max(x), len = input$bins + 1)
    hist(x, breaks = bins, col = 'darkgray')
  })
}

ui <- fluidPage(
  titlePanel("Old Faithful Geyser Data"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins","Number of bins:",
        min = 1, max = 50, value = 30)),
    mainPanel(plotOutput("distPlot"))
))

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



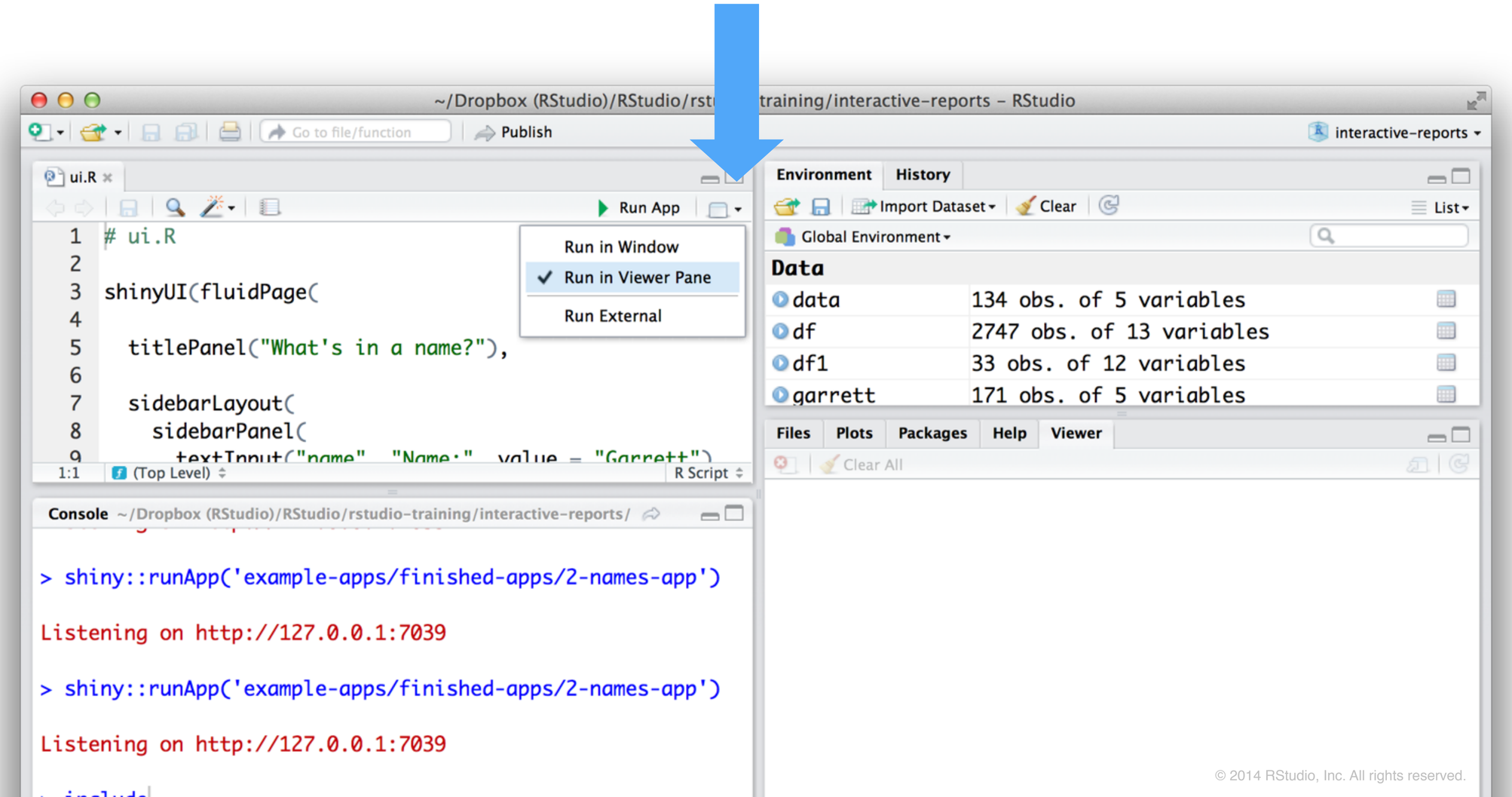
```
# server.R
library(shiny)
function(input, output) {
  output$distPlot <- renderPlot({
    x    <- faithful[, 2]
    bins <- seq(min(x), max(x), len = input$bins + 1)
    hist(x, breaks = bins, col = 'darkgray')
  })
}
```

```
# ui.R
fluidPage(
  titlePanel("Old Faithful Geyser Data"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins","Number of bins:",
        min = 1, max = 50, value = 30)),
    mainPanel(plotOutput("distPlot"))
))
```

Launch an app



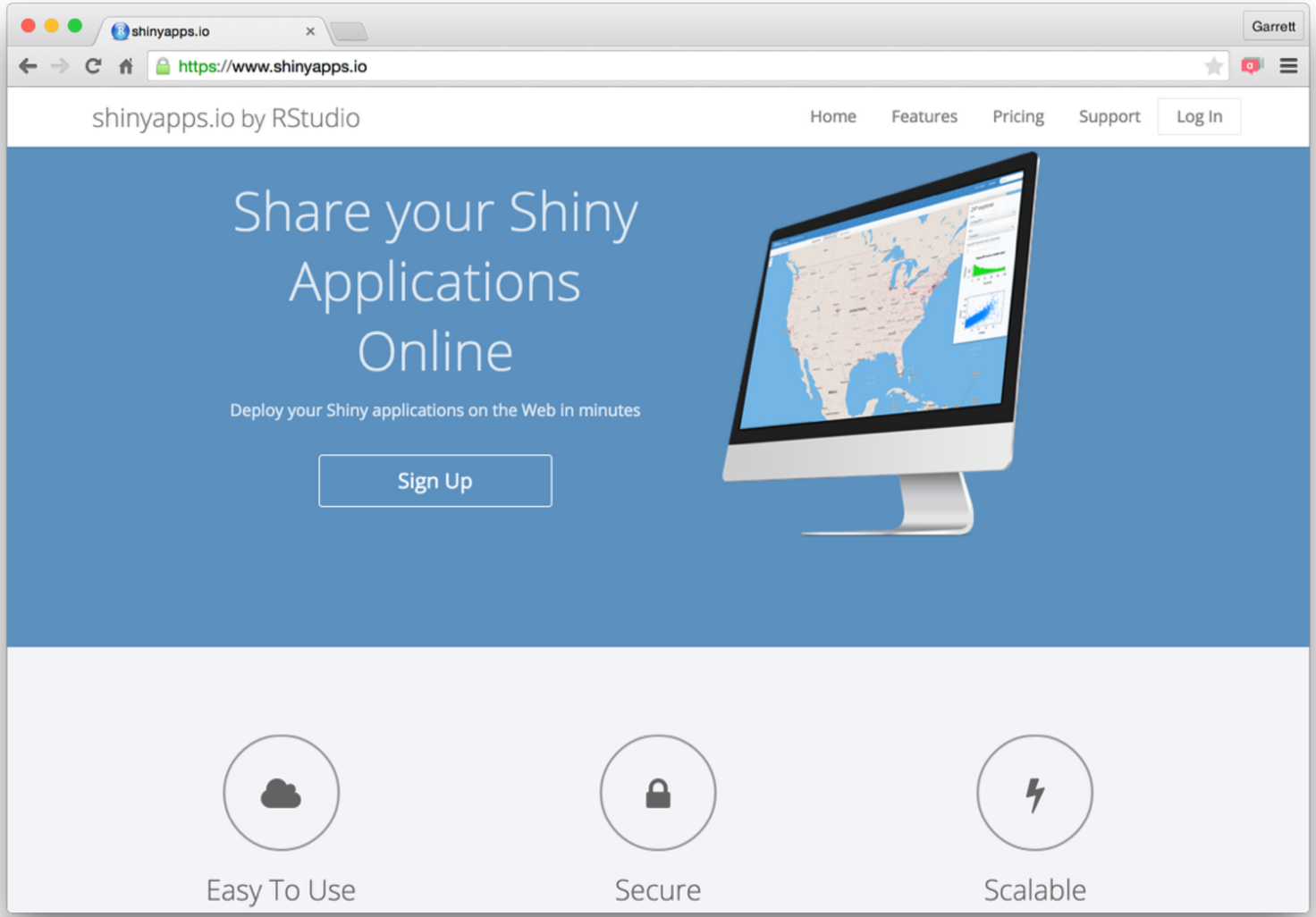
Display options



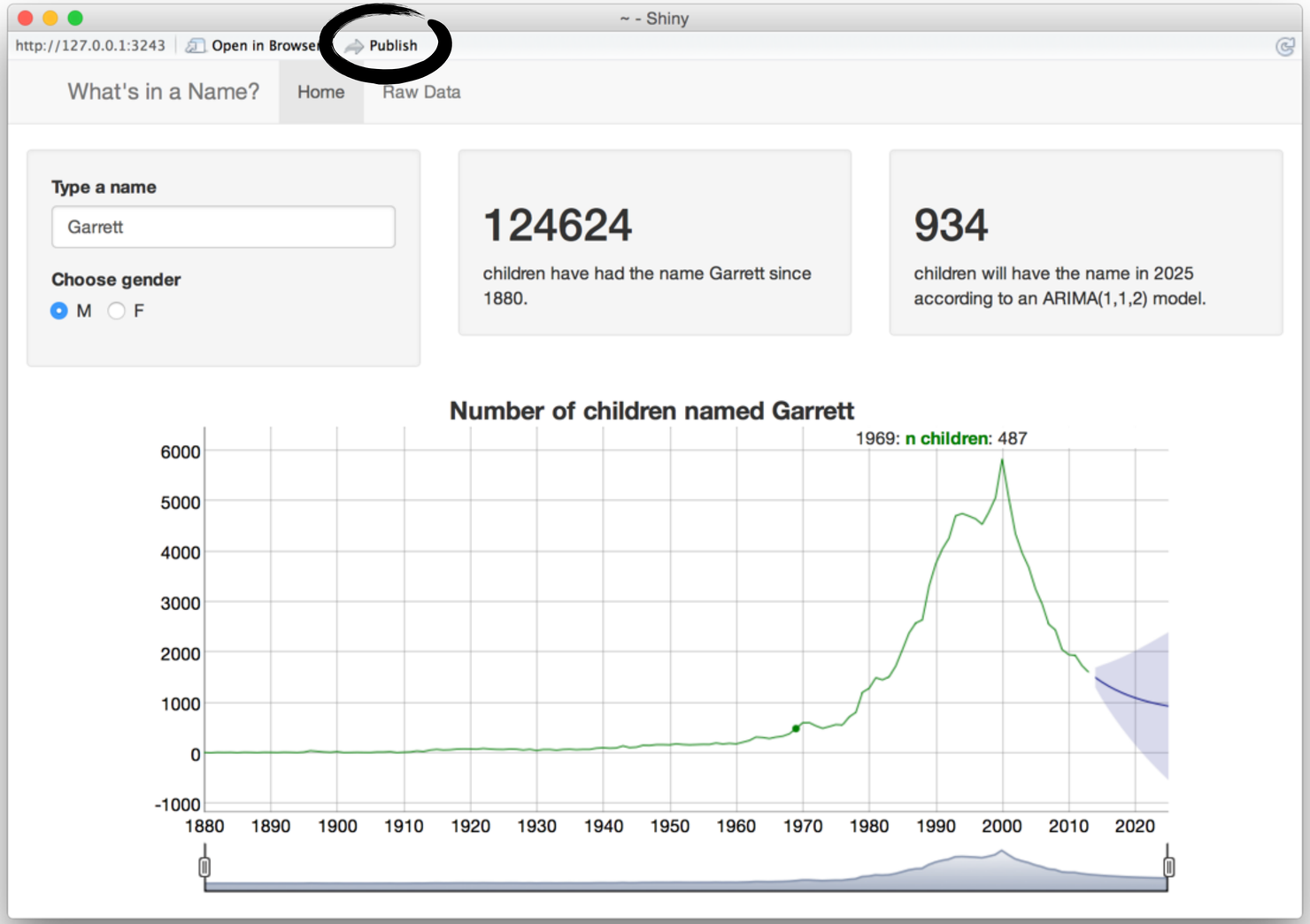
shinyapps.io

Hassle-free cloud hosting for Shiny

shinyapps.io

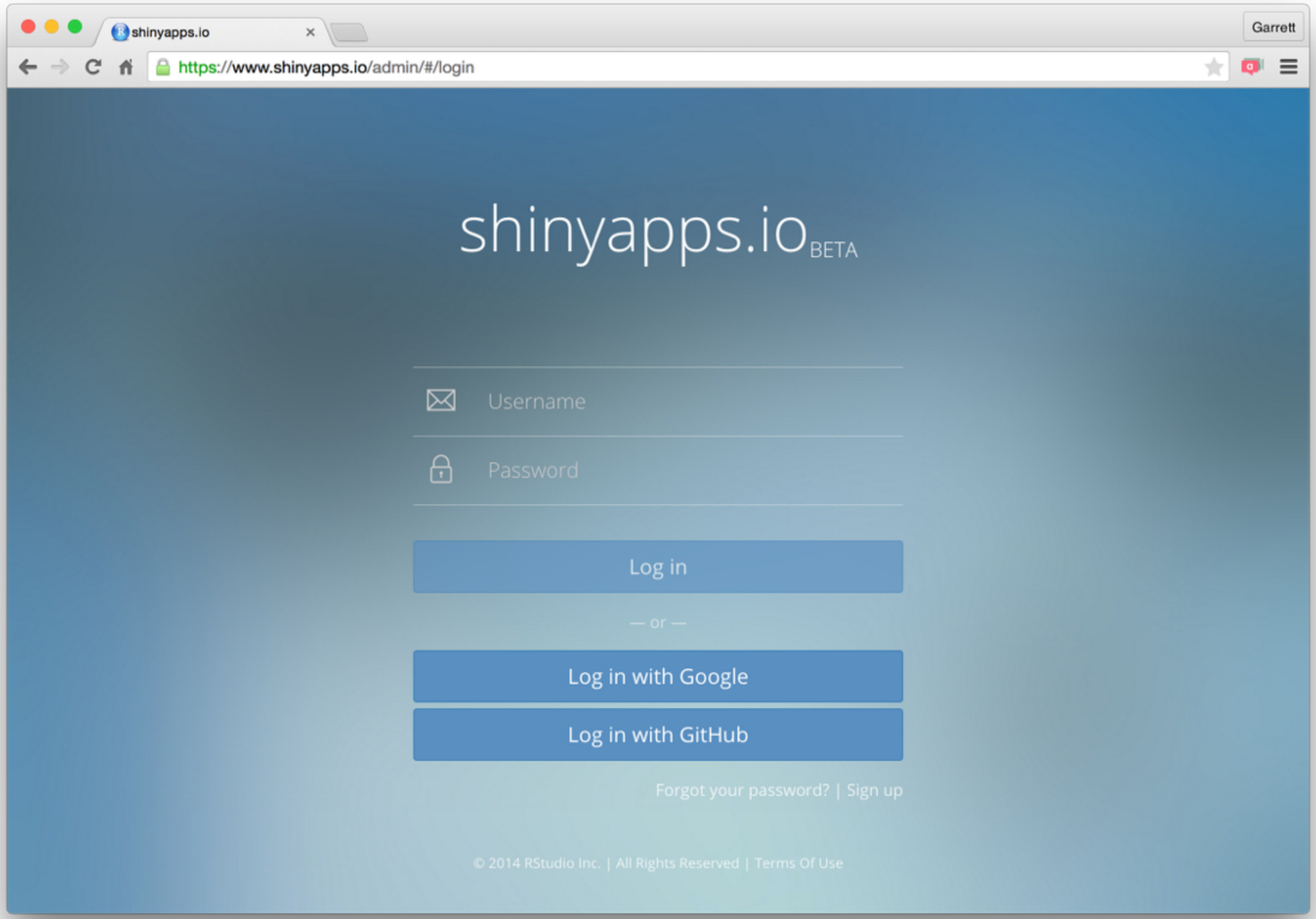


Deploy to web with one click



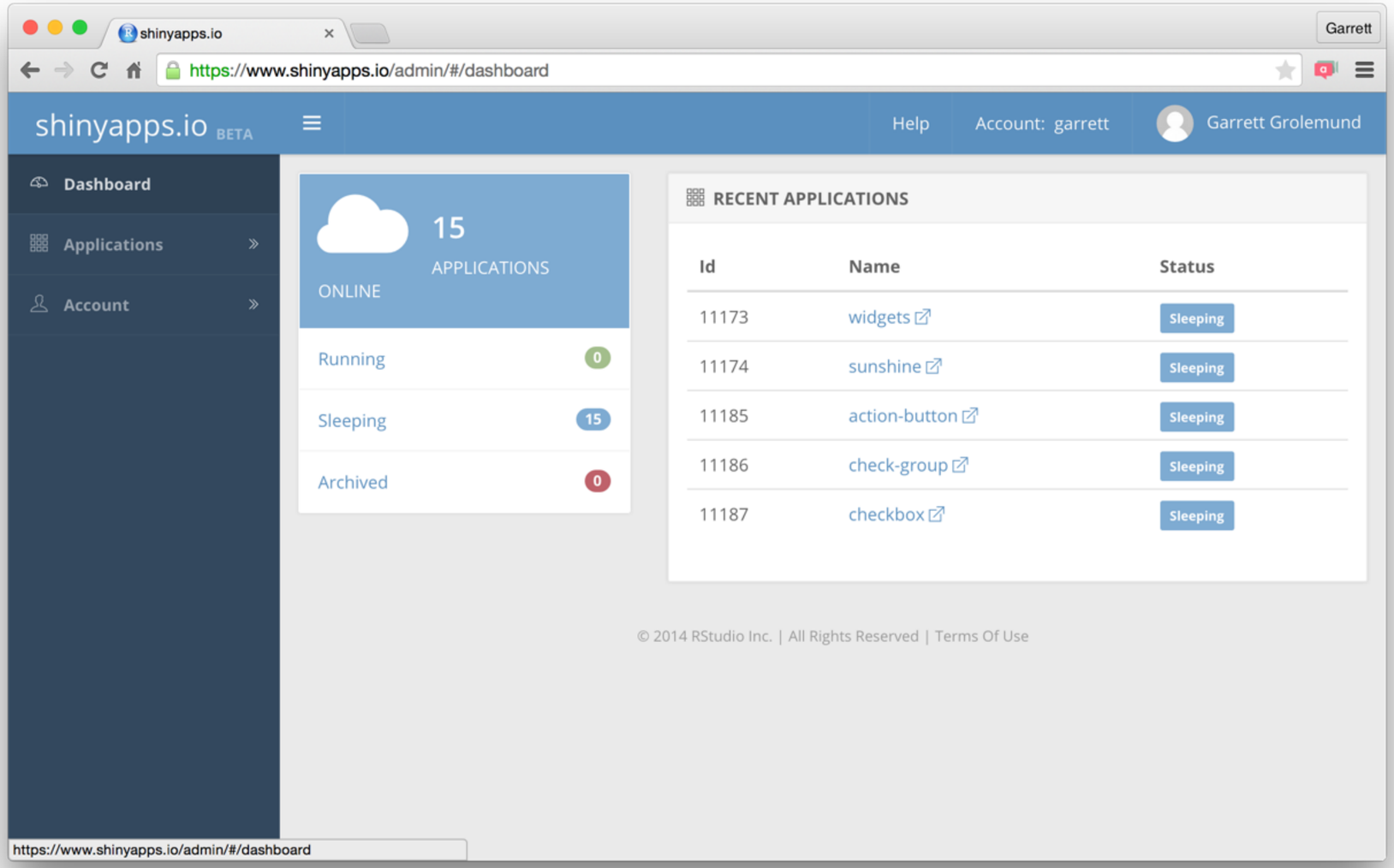
Hassle-free cloud hosting for Shiny

shinyapps.io



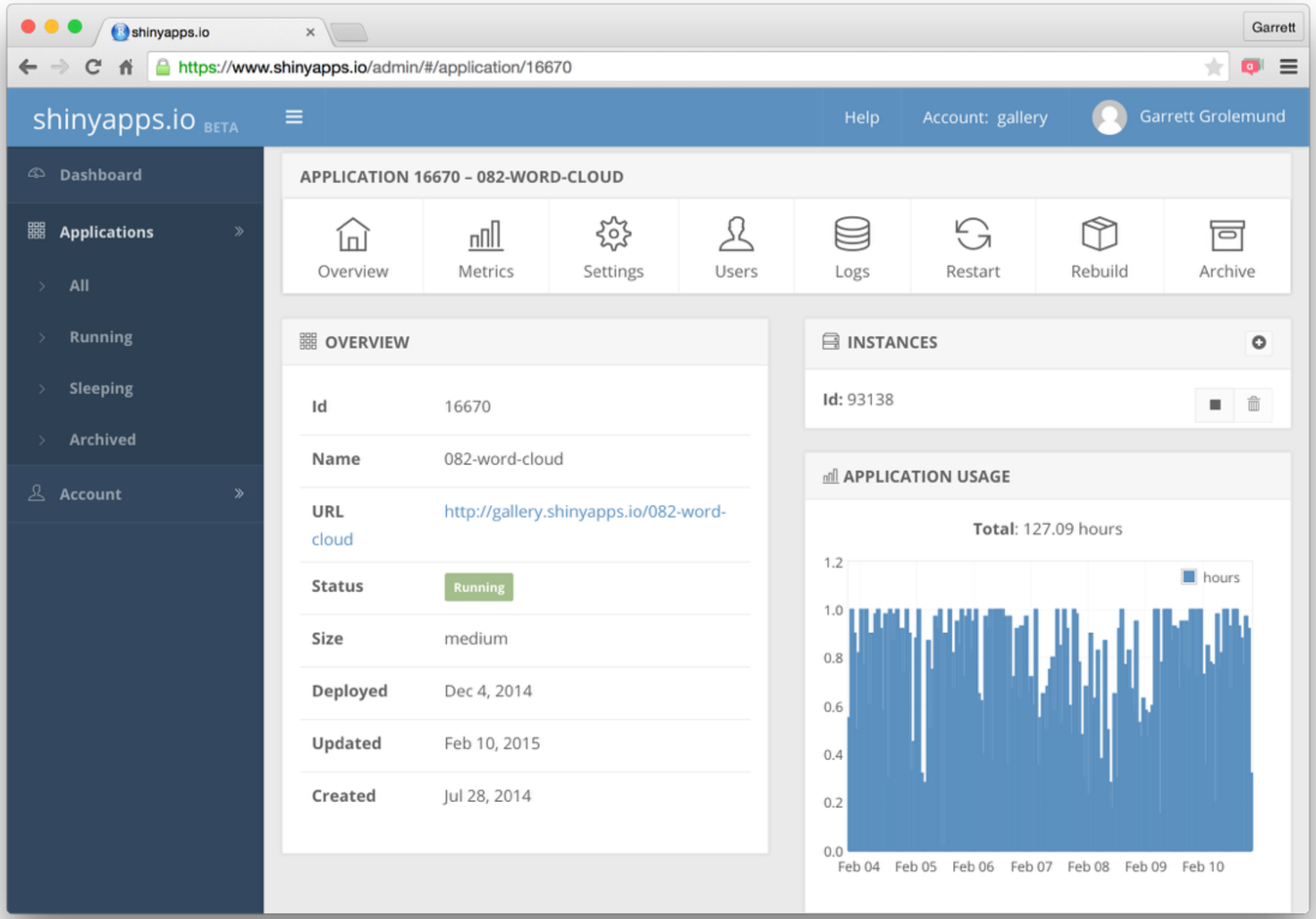
Hassle-free cloud hosting for Shiny

shinyapps.io



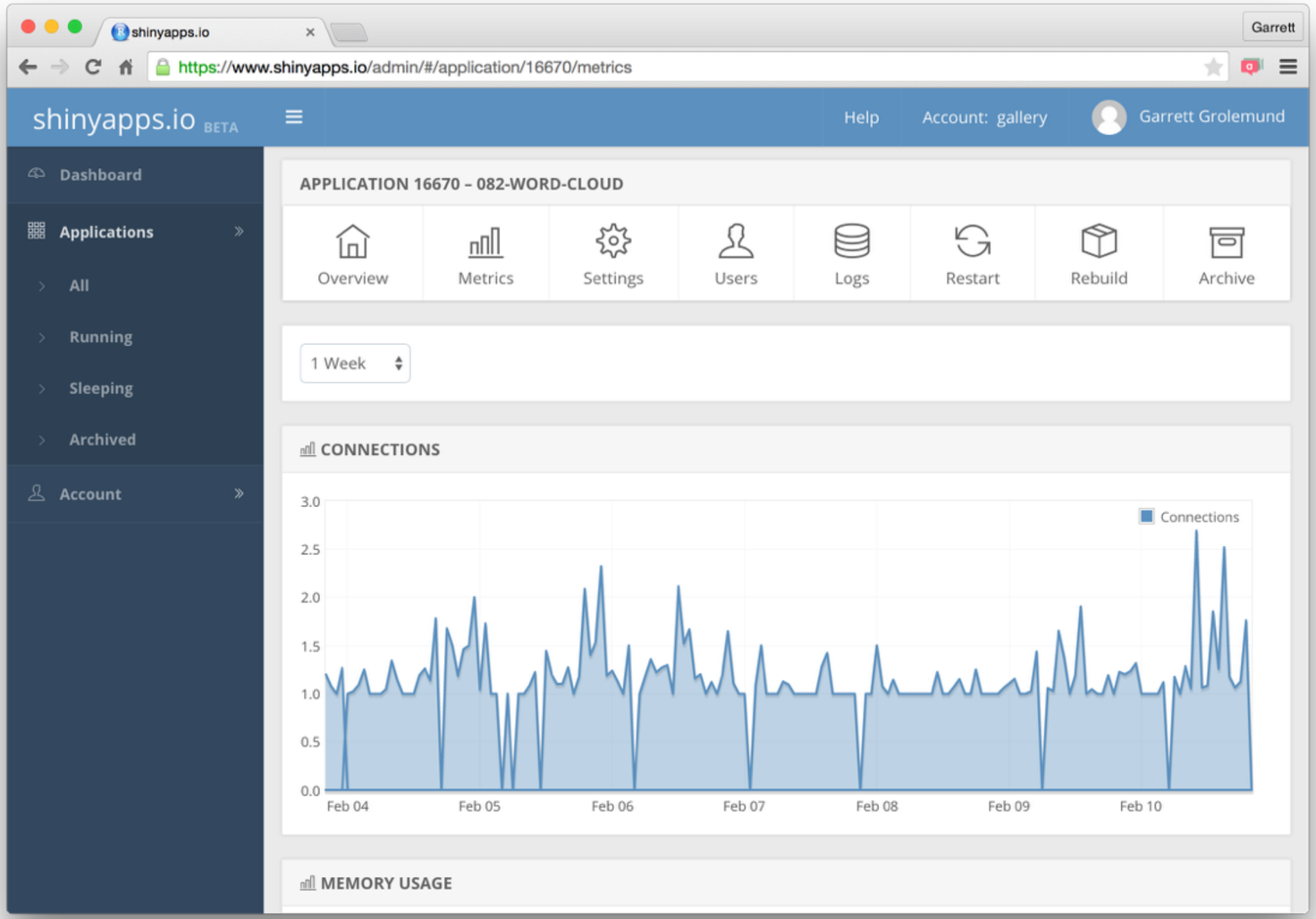
Hassle-free cloud hosting for Shiny

shinyapps.io



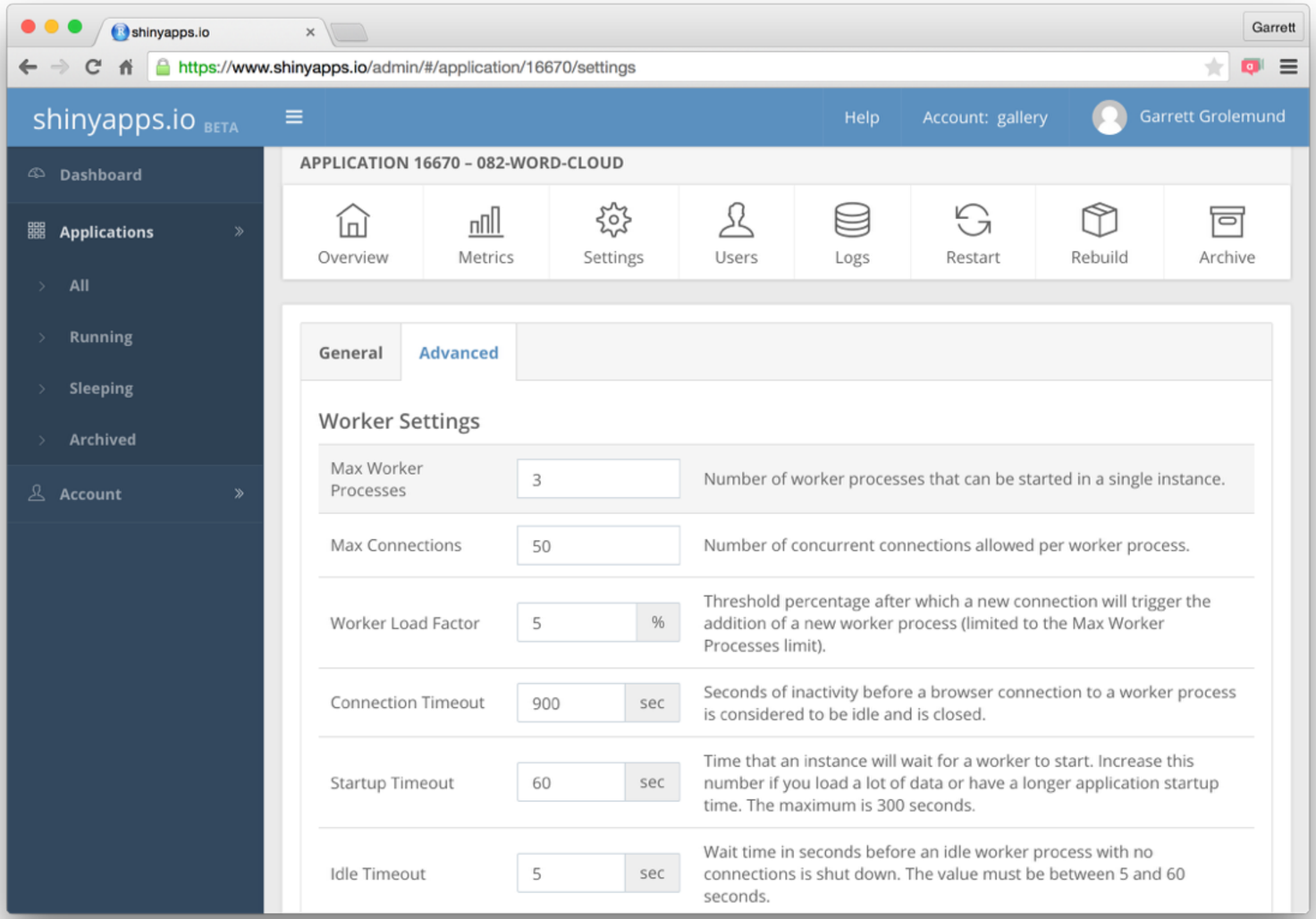
Hassle-free cloud hosting for Shiny

shinyapps.io



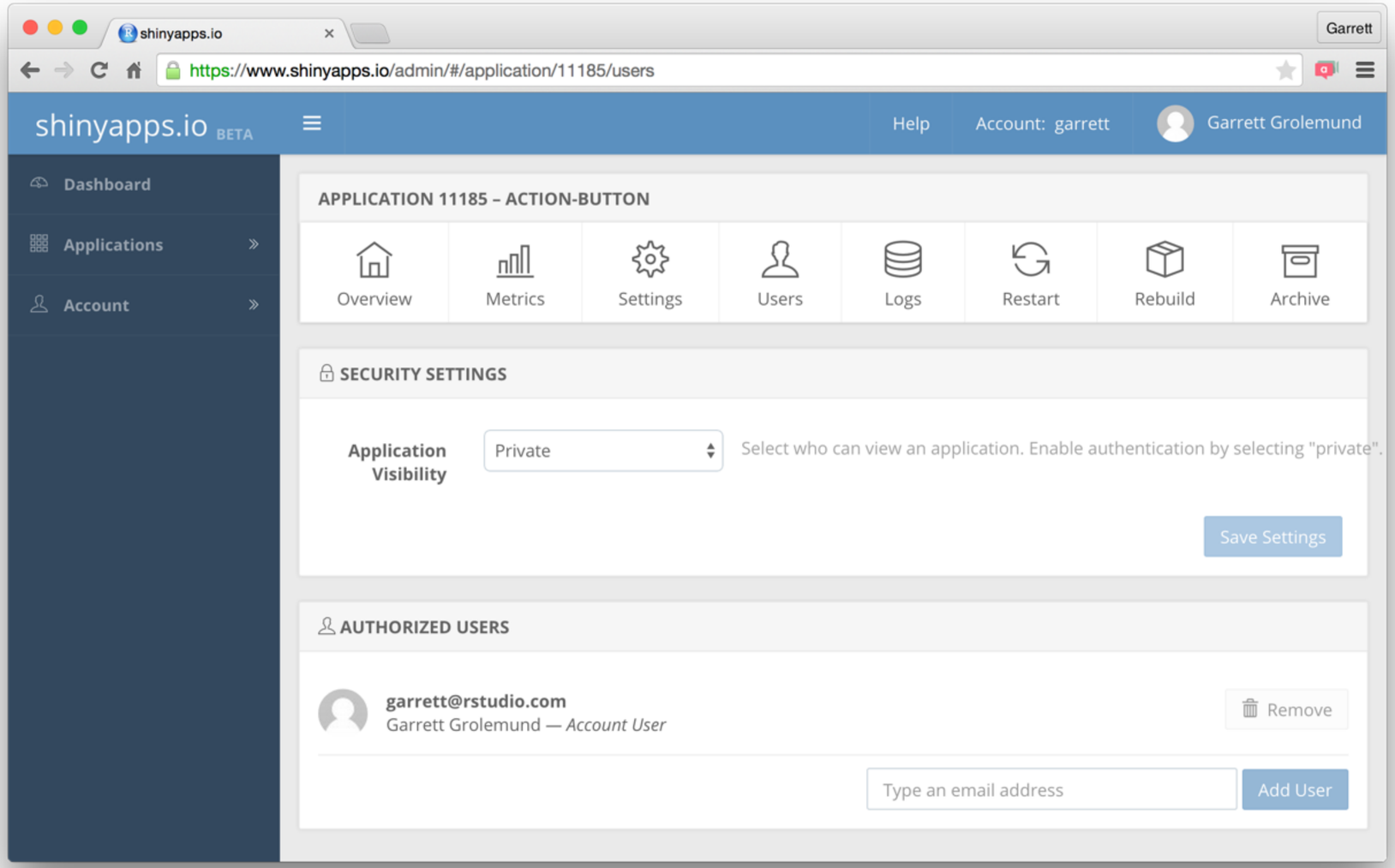
Hassle-free cloud hosting for Shiny

shinyapps.io



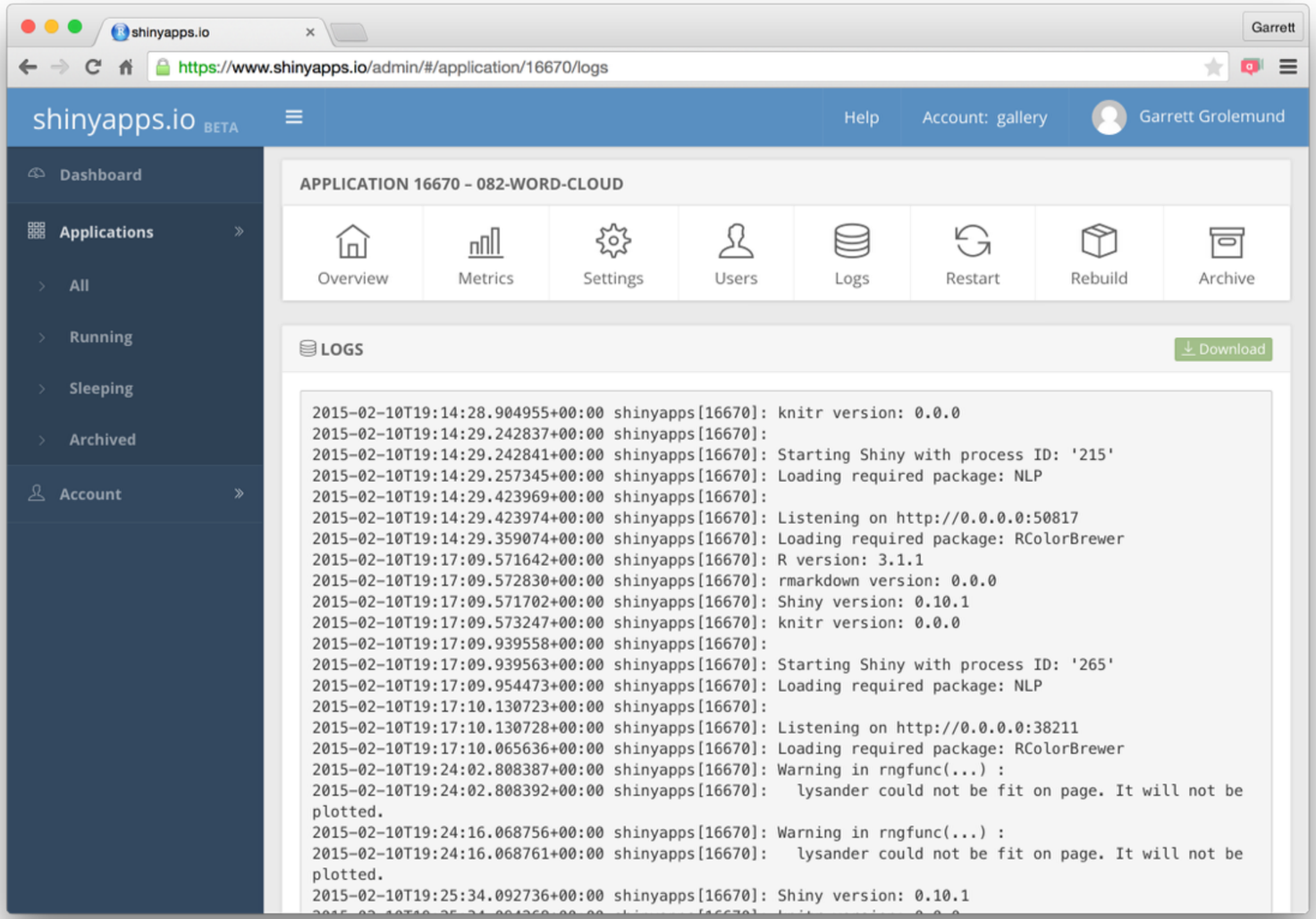
Hassle-free cloud hosting for Shiny

shinyapps.io



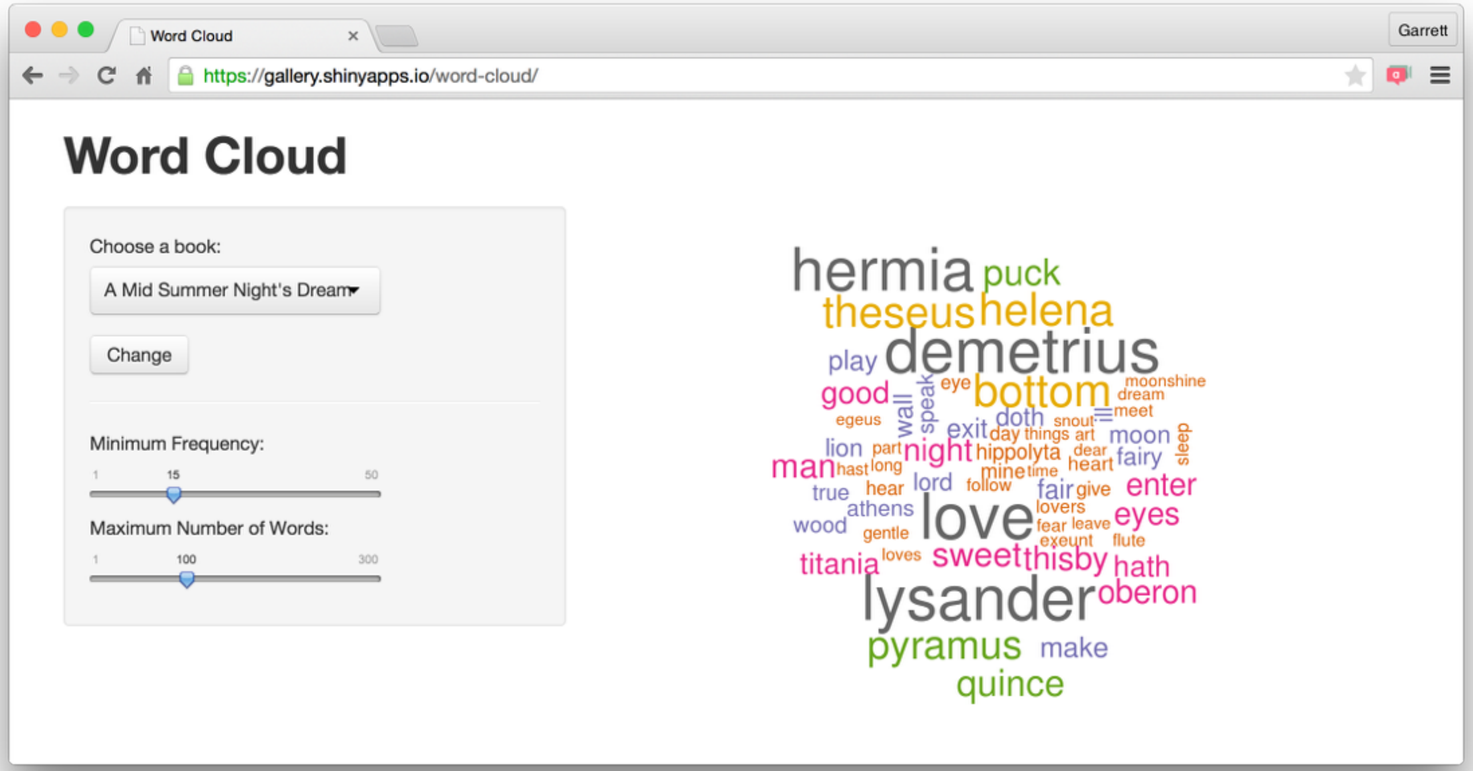
Hassle-free cloud hosting for Shiny

shinyapps.io



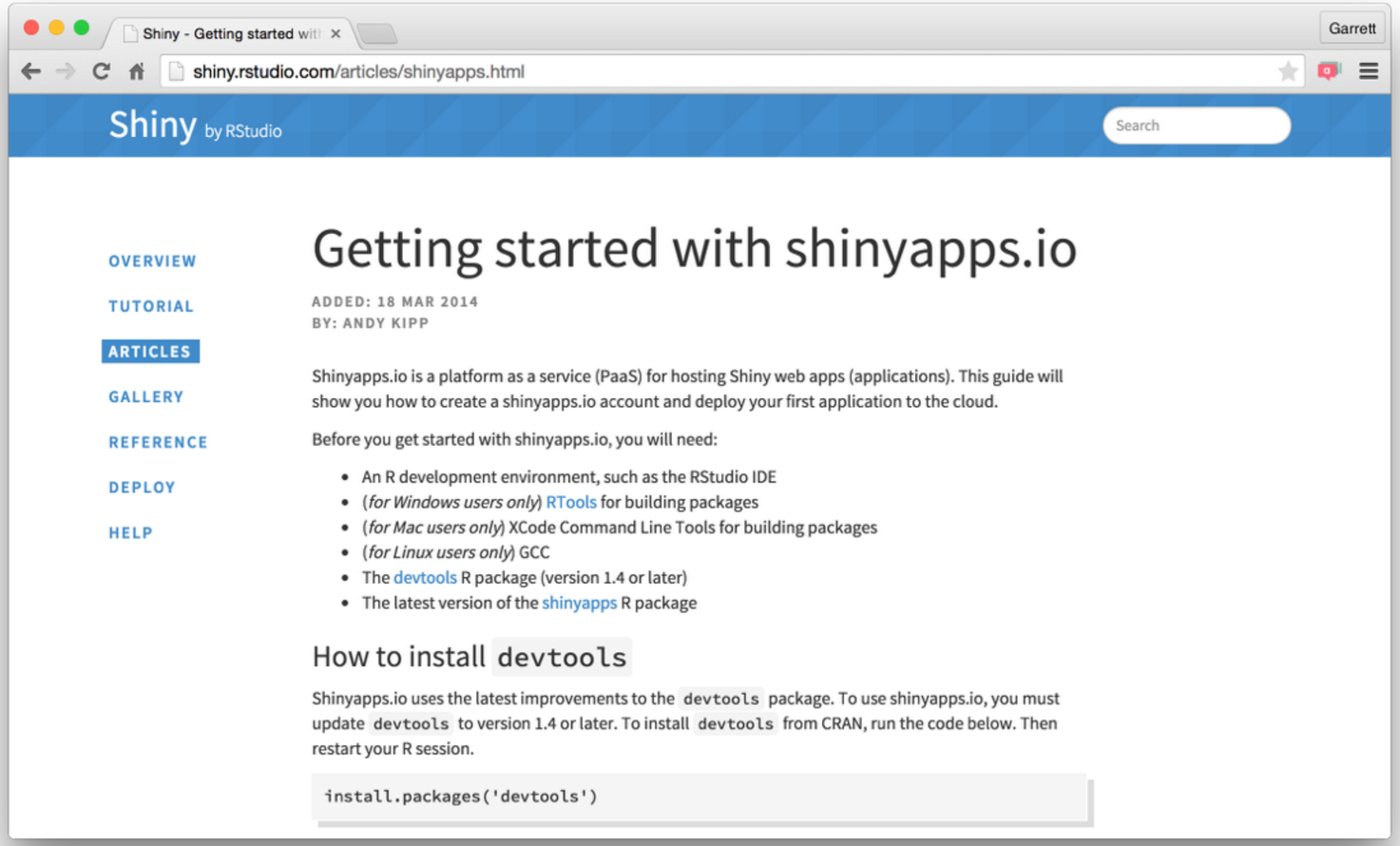
Each app hosted at its own URL

gallery.shinyapps.io/word-cloud



Getting started guide

shiny.rstudio.com/articles/shinyapps.html



Build a server

Shiny Server

A back end program that builds a web server specifically designed to host Shiny apps

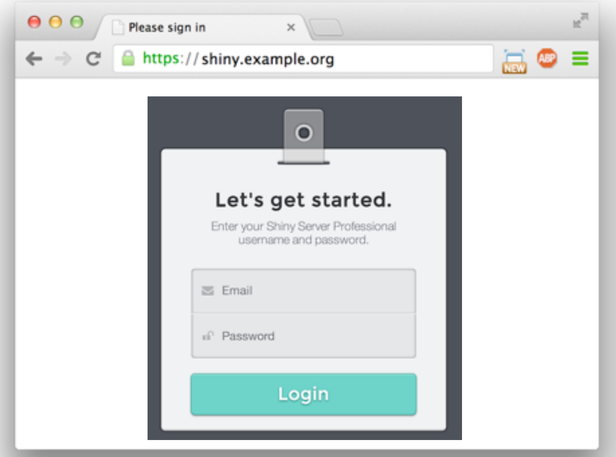
- Each app is hosted at its own URL
- Can deploy to internet, or within a controlled environment
- Starts app when user visits, closes app when user leaves
- Runs on a linux server

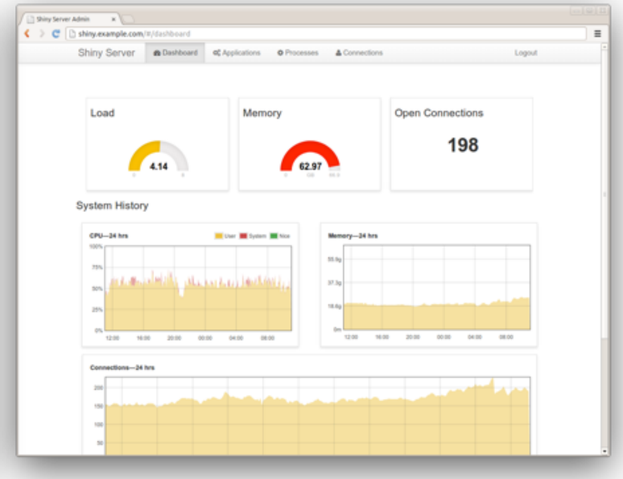
http://shiny.rstudio.com/articles/shiny-server.html

Shiny Server Pro

www.rstudio.com/products/shiny-server-pro/

- **Secure access** - LDAP, GoogleAuth, SSL, and more
- **Performance** - fine tune at app and server level
- **Management** - monitor and control resource use
- **Support** - direct priority support





45 day evaluation free trial

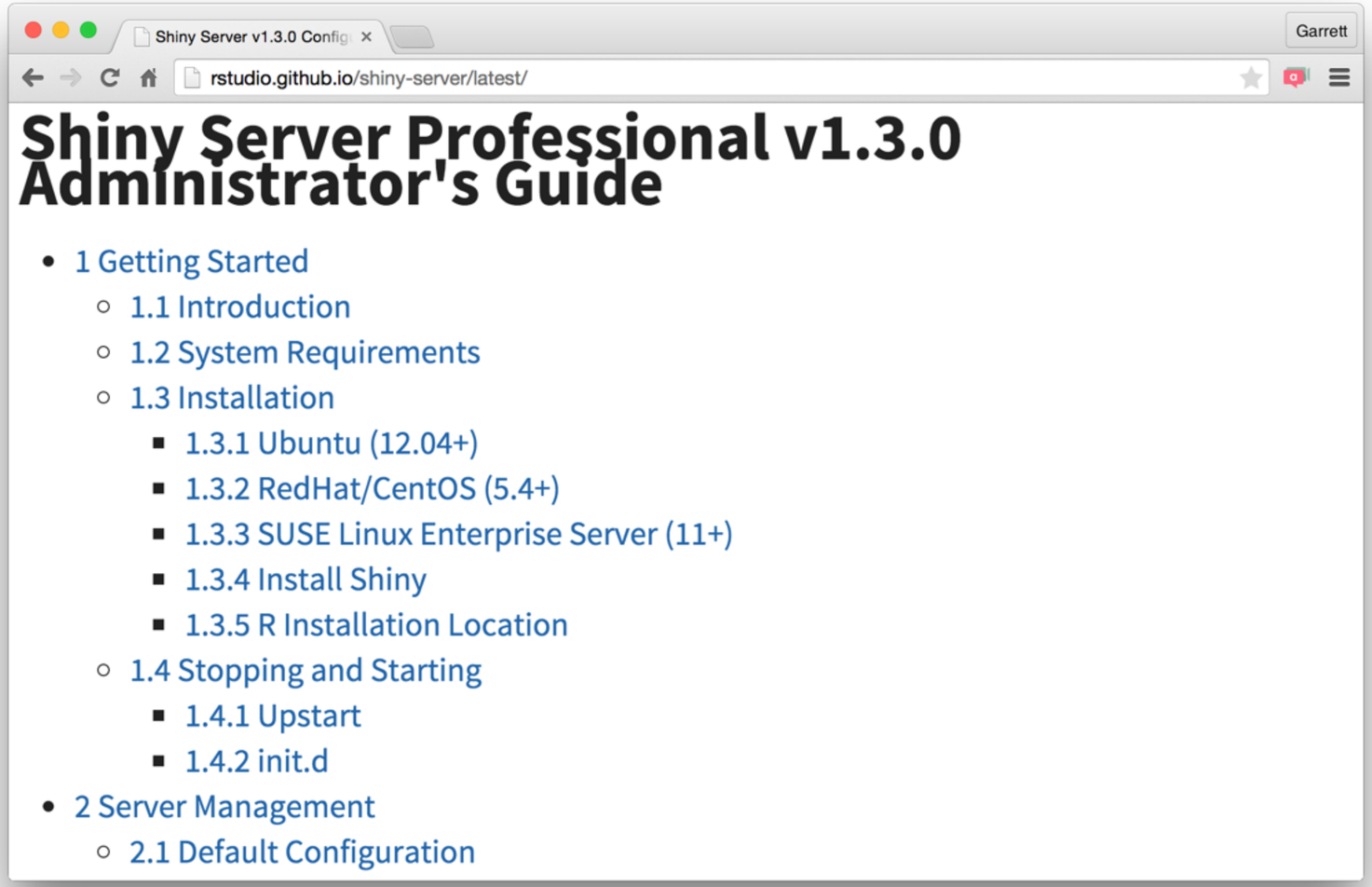
http://rstudio.com/shiny/server

		Open Source Edition	Professional Edition
General	Deploy Shiny applications to the Internet	•	•
	Move computation close to the data	•	•
	Host multiple applications on a single server	•	•
	Deploy Shiny applications behind firewalls	•	•
	Custom page templates	•	•
Security and authentication	Password file authentication		•
	LDAP and Active Directory authentication		•
	Google authentication (OAuth2)		•
	PAM authentication & sessions		•
	Group based authorization		•
	SSL support		•
Tuning and scaling	Scale applications across multiple processes		•
	View and manage active sessions		•
	Allocate resources on a per application basis		•
	Define application concurrency limits		•
Server monitoring	System performance and resource metrics		•
	Per application performance and resource metrics		•
	Application usage metrics		•
	Customizable health check end point		•

* For volume discounts, OEMs, or additional capacity for larger audiences please email us at sales@rstudio.com.

Shiny Server (Pro) Admin Guide

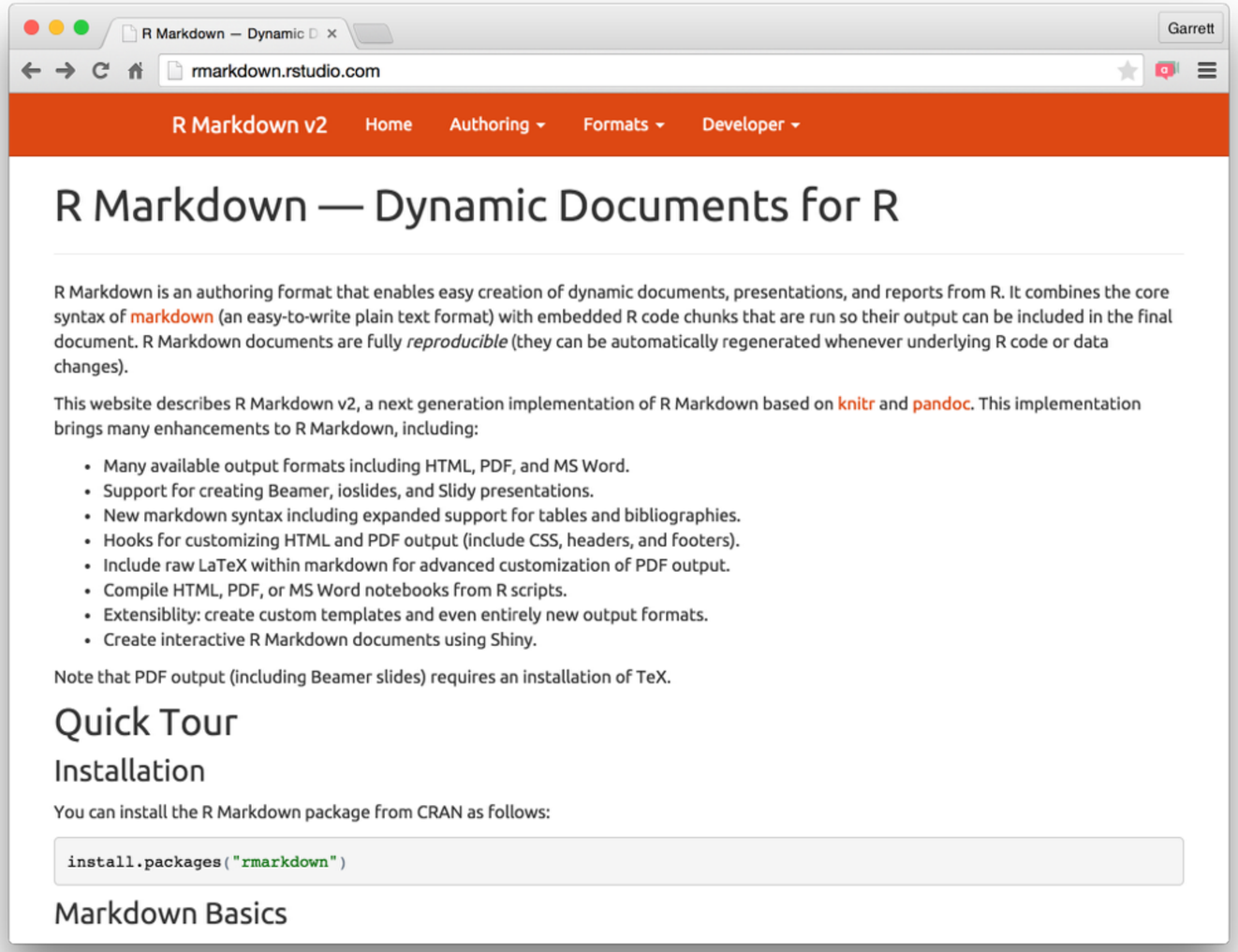
rstudio.github.io/shiny-server/latest/



How to learn more

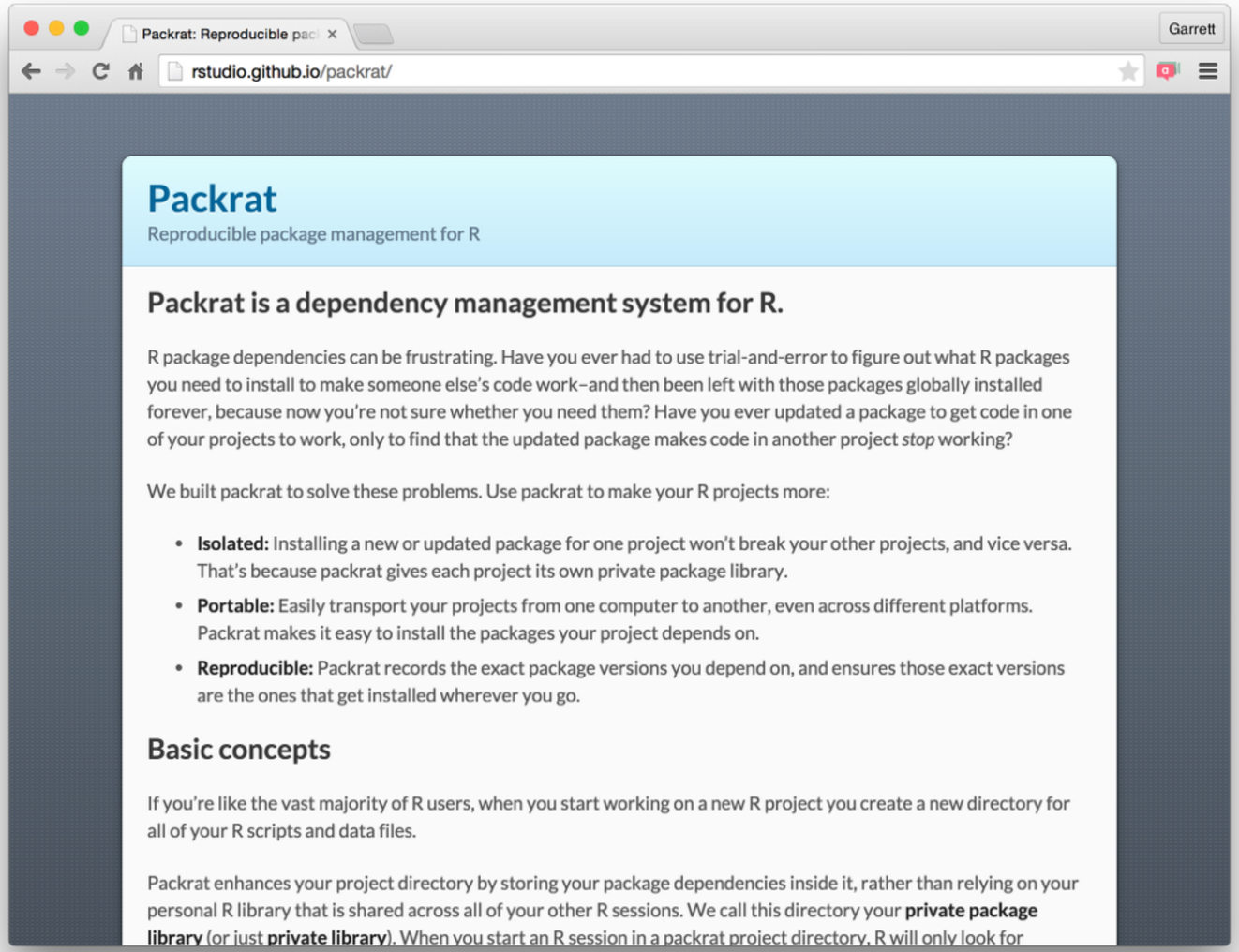
The R Markdown Development Center

rmarkdown.rstudio.com



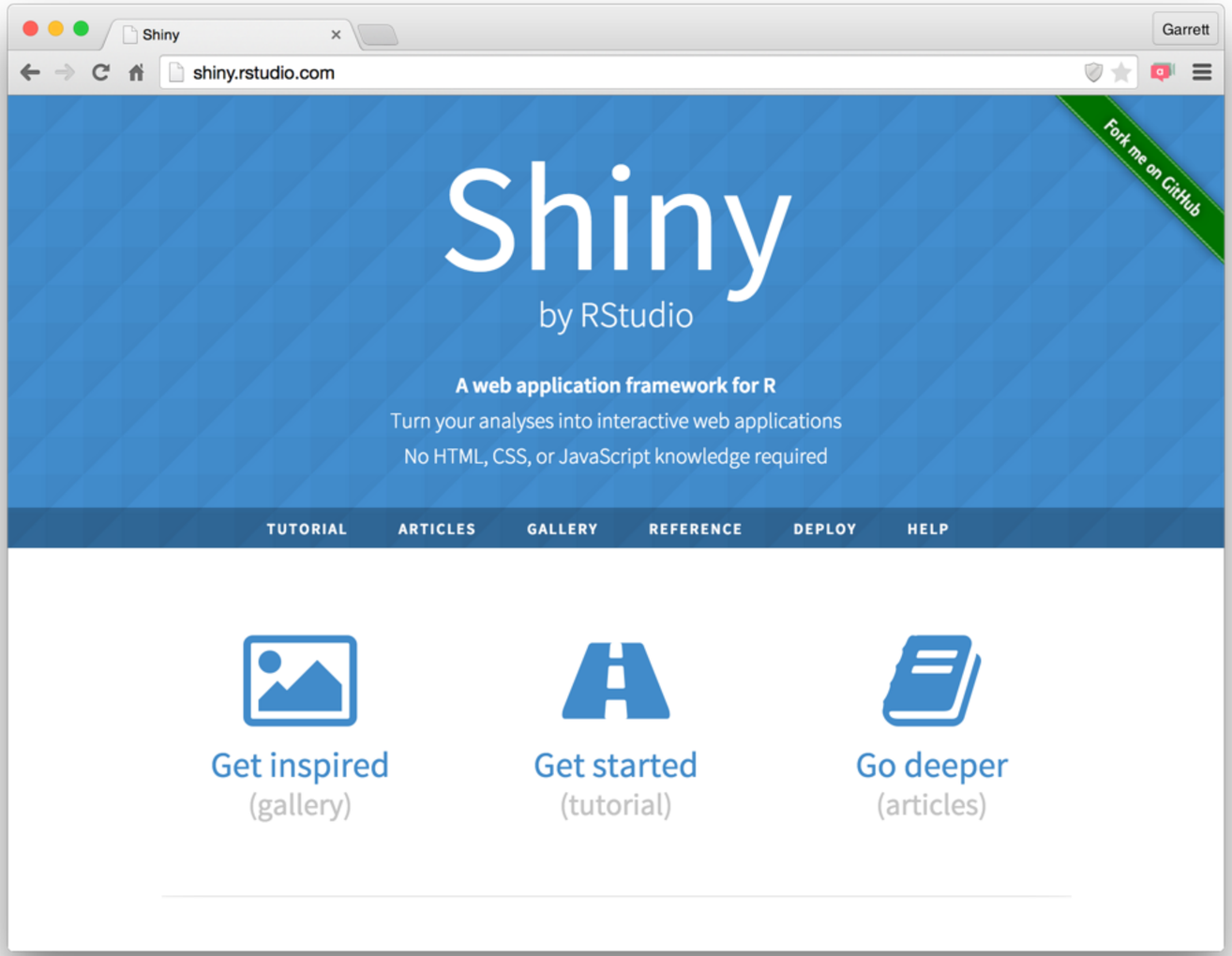
The Packrat Development Center

rstudio.github.io/packrat/



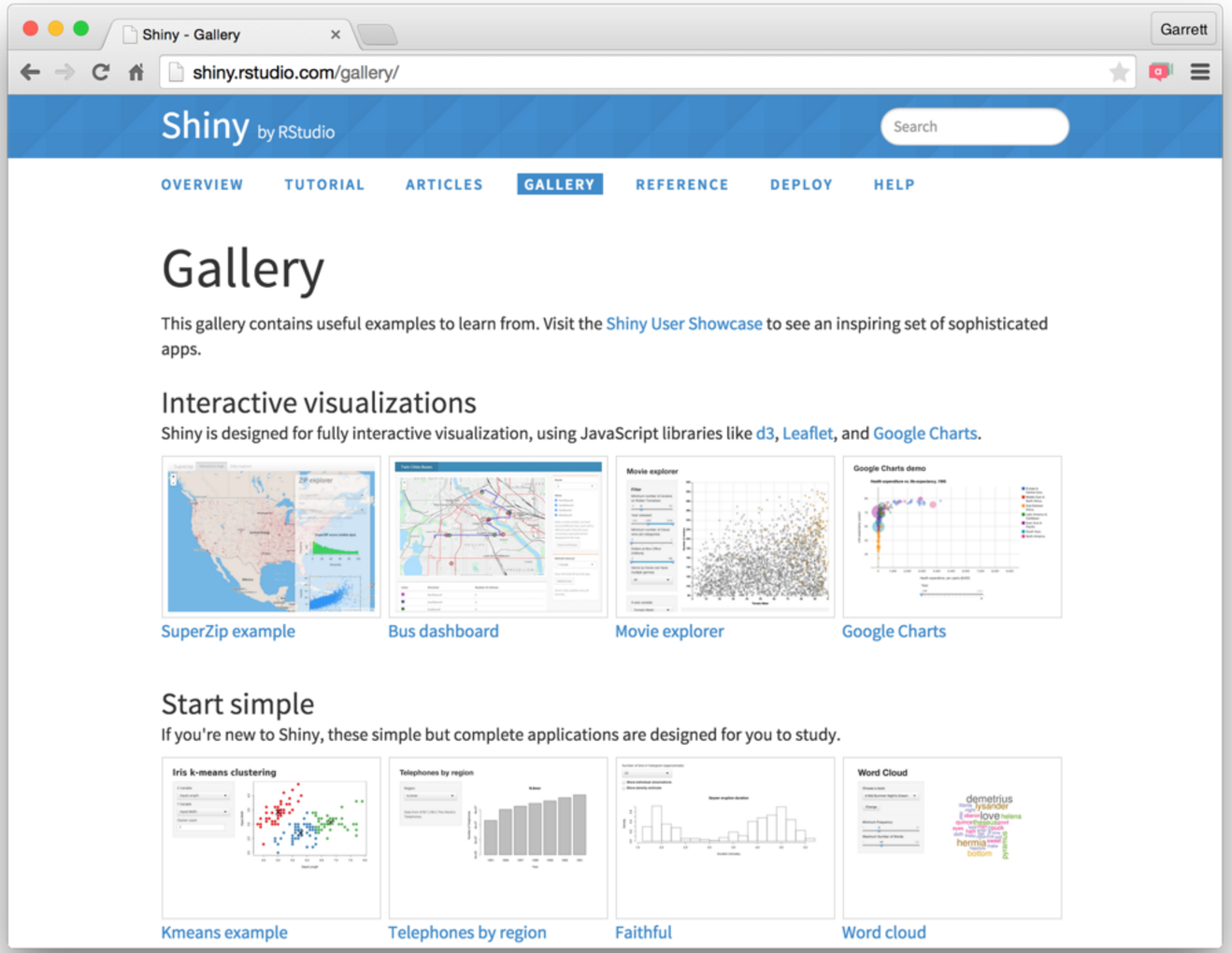
The Shiny Development Center

shiny.rstudio.com



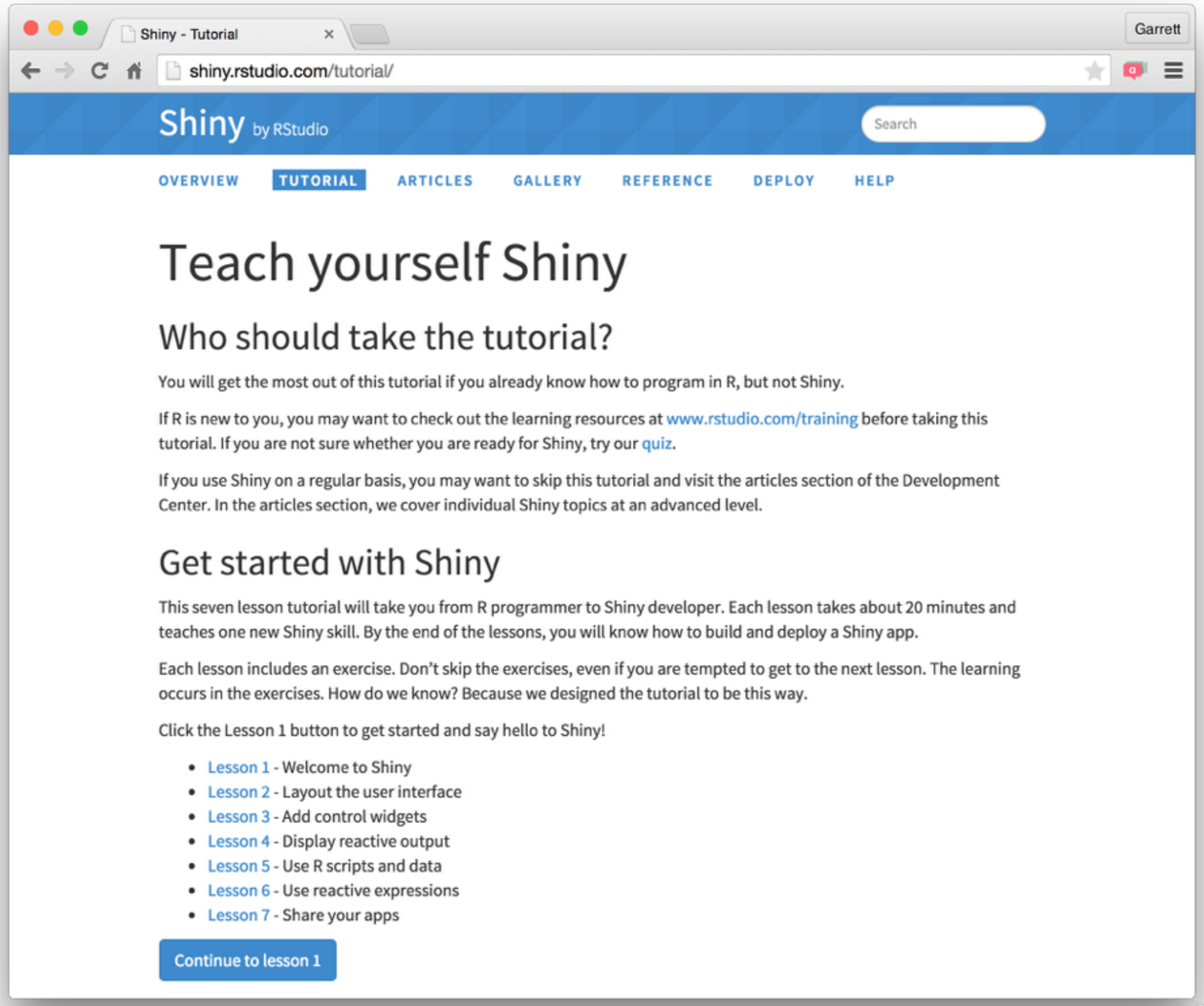
The Shiny Development Center

shiny.rstudio.com/gallery



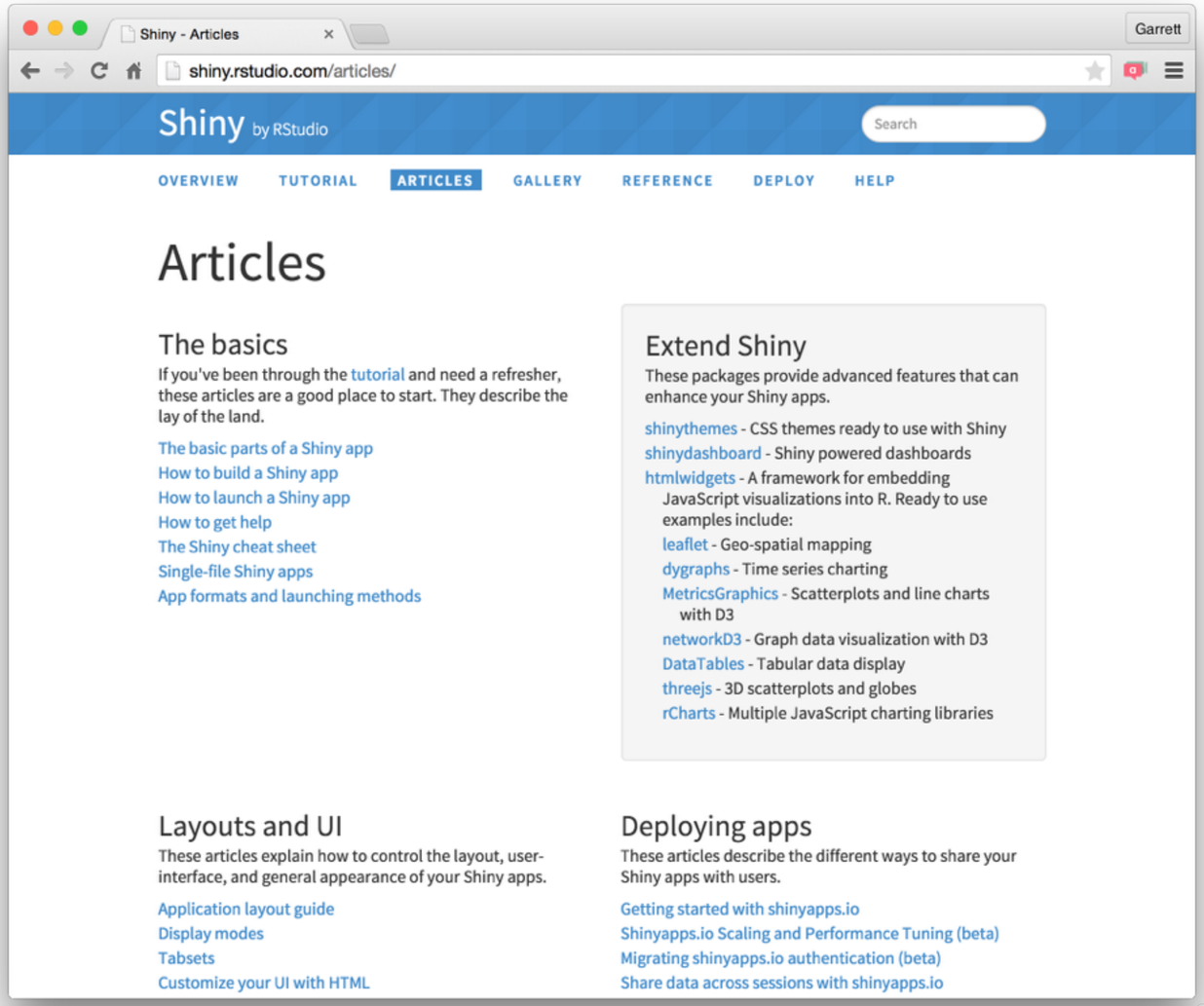
The Shiny Development Center

shiny.rstudio.com/tutorial



The Shiny Development Center

shiny.rstudio.com/articles



Thank you

Reactivity with Shiny

Slides at: bit.ly/EARL2015-Reports

Much more at: shiny.rstudio.com

R Markdown at: rmarkdown.rstudio.com

Packrat at: rstudio.github.io/packrat/