Shiny WebApp - The Basics

Fu-Sheng Chou, MD, PhD

This month, we will start the journey into Shiny WebApp development using R and the RStudio integrated development environment. I hope you have had R and RStudio installed or that you could register for an account at RStudio Cloud to get started. Now, let us open RStudio to get started (you do not have to open R).

In order to create shiny apps, you will need to install the package:

> install.packages("shiny")

After installation, you will be able to create a new Shiny WebApp file by clicking on File > New File > Shiny Web App… (Figure 1). A dialogue pops up asking for the Application name (I called it NAS) and the directory (I have it under R/Projects/) where the application will reside (Figure 2). After hitting Create, a Shiny WebApp file with template codes will show up on the left panel (Figure 3). The tab says "app.R", which is the name of the Shiny WebApp code file. Note the right lower panel with the tab Files showing a folder named "NAS" was automatically created (Figure 3 green circle).

"Briefly speaking, a WebApp contains a user interface to interact with the user, a background server that takes the input values from the user interface for computation, followed by outputting the computed values back to the user interface."

Inside the "NAS" folder is the file app.R which was opened on the left side.

The code structure for a Shiny WebApp has two parts:

1. The user interface: **ui <- fluidPage()**
2. The server: **server <- function(input, output) {}**

At the end of the template is a function **ShinyApp()** with two arguments (ui, server) that is used to run the Shiny WebApp.

Briefly speaking, a WebApp contains a user interface to interact with the user, a background server that takes the input values from the user interface for computation, followed by outputting the computed values back to the user interface. The input and output arguments in the server function are actually two "containers" that store the widgets to interact with the user (Figure 4).

Now, hit Run App (Figure 3 red circle) and see what happens. Compare the WebApp and the code:

1. **Code**: There is a titlePanel() function with a title in the character data type (flanked by double quotations).

   App: The title is displayed in the left upper corner.

2. **Code**: The sidebarLayout() function allows you to build the WebApp with a two-column layout: a sidebar on the left and a main display area on the right. Correspondingly, you see that, inside the sidebarLayout() function, there are two functions: sidebarPanel() and mainPanel() that are used to build the widgets and to display the output.

   1. sidebarPanel(): Inside this function is a widget function called sliderInput(), with various arguments in it, as listed in Table 1:
   2. mainPanel(): Inside this function is an output function for plot called plotOutput(). There is only one argument: the unique character id "distPlot" assigned, so R knows to refer to this specific output.

   App: The left column has a slider with the corresponding label; the minimum number is 1, and the maximum number is 50, and the default value is 30. The right side is a plot.

"There are three major types of layout that Shiny supports: plain layout, the sidebar layout, and the dashboard layout."

Now let us play with the slider left or right to see what happens to the plot on the right. When you move the slider, the bin number of the histogram on the right changes accordingly!

User Interface

There are three major types of layout that Shiny supports: plain layout, the sidebar layout, and the dashboard layout.

The structure of a plain layout (Figure 5A) is as follow:

```
ui <-   fluidPage(
        titlePanel(),
        input widgets and output elements
        )
```

A title and a body

The structure of a sidebar layout[1] (Figure 5B) is as follow:

```
ui <-   fluidPage(
```

titlePanel(),

sidebarPanel(*input and output widgets*, width = 4),

mainPanel(*input and output widgets*, width = 8),

position = c("left", "right")

)

A title, a sidebar panel, and the main panel

The position argument is used to determine the sidedness of the sidebar. The default is *left*, which is the first element of the vector. Any given webpage is divided into 12 invisible vertical strips with equal width, and the default widths for the side panel and the main panel are 4 and 8 strips, respectively. You can change that by changing the width argument inside the sidebarPanel() and mainPanel() functions.

The structure of a dashboard layout[2] (Figure 5C) is as follow:

ui <- dashboardPage(

dashboardHeader(),

dashboardSidebar(),

dahsboardBody()

)

A header, a sidebar, and a body

> ***"You may play with the widget in the gallery to get an idea of the value corresponding to user action."***

Note that the *shinydashboard* package needs to be installed [by typing **install.packages("shinydashboard")** in the console]and loaded [by typing **library(shinydashboard)** in the console] before you can use a dashboard layout.

Inside each panel are input widgets and output elements. The basic shiny widget gallery is available in Ref 3. You may play with the widget in the gallery to get an idea of the value corresponding to user action. The values are stored insider the input container. As in the above example, the slider value is stored in input$bins. In the server function, the values resulting from user action are retrieved by accessing the input container's bins compartment (hence input$bins, please refer to the January 2021 column article for details).

For output, in addition to plotOutput() mentioned above, some additional examples of the common output types are listed in Table 2.

Can the output be input widgets? Absolutely! Use uiOutput() to generate new input widgets based on the value obtained from an input widget.

Server Function

An important concept in Shiny WebApp is **reactivity** (Figure 4). That is, when the user changes the input value(s), the server repeats the computation based on the new value(s) and updates the output(s). With regards to the NAS WebApp we are going to develop, upon *entering the NAS score for each item, the total NAS score is updated correspondingly and immediately.*

Okay, so how is the histogram (or the output in general) generated? To answer this question, we have to turn our attention to the server function. The server object has a function with two arguments called input and output. The structure is as such:

server <- function(input, output) {

compute and send the results to the output

}

We will not go into details about how to plot a histogram now. I just want to point out how to send the computed results to the output. If you remember from above, inside the mainPanel() function, which stores all the information for the right column, there is a plotOutput() function with a single argument "distPlot" for the output id. We also said that input and output are containers. Now, inside the server function, you see a structure like this:

output$distPlot <- renderPlot({*codes to generate the plot*})

This code means that the renderPlot() function contains a bunch of computation work inside the curly brackets, and the results from the computation (in this case, plotting a histogram) will be inside the *distPlot* compartment of the *output* container.

plotOutput() in the *user interface* and renderPlot() in the *server function* have to match each other; otherwise, the communication will result in an error message. Some examples are listed in Table 2.

> ***"Next month, we will discuss the detailed web page layout and various elements (widgets, text, lines, space, etc.) that are available to you in designing your user interface."***

To summarize, we discussed the code structure for a shiny app: the **user interface** and the **server function**. There are input and output in the user interface, which are containers to store values from the corresponding input widgets and computation results for display, respectively. In the server function, computation occurs based on input values, and the results are presented to the output elements. The NAS WebApp codes are available here (https://neonatologytoday.org/datascience/NAS.html) for those readers who would like to apply what we learned here to a real-life Neonatology-relevant example. Next month, we will discuss the detailed web page layout and various elements (widgets, text, lines, space, etc.) that are available to you in designing your user interface.

Figure 1

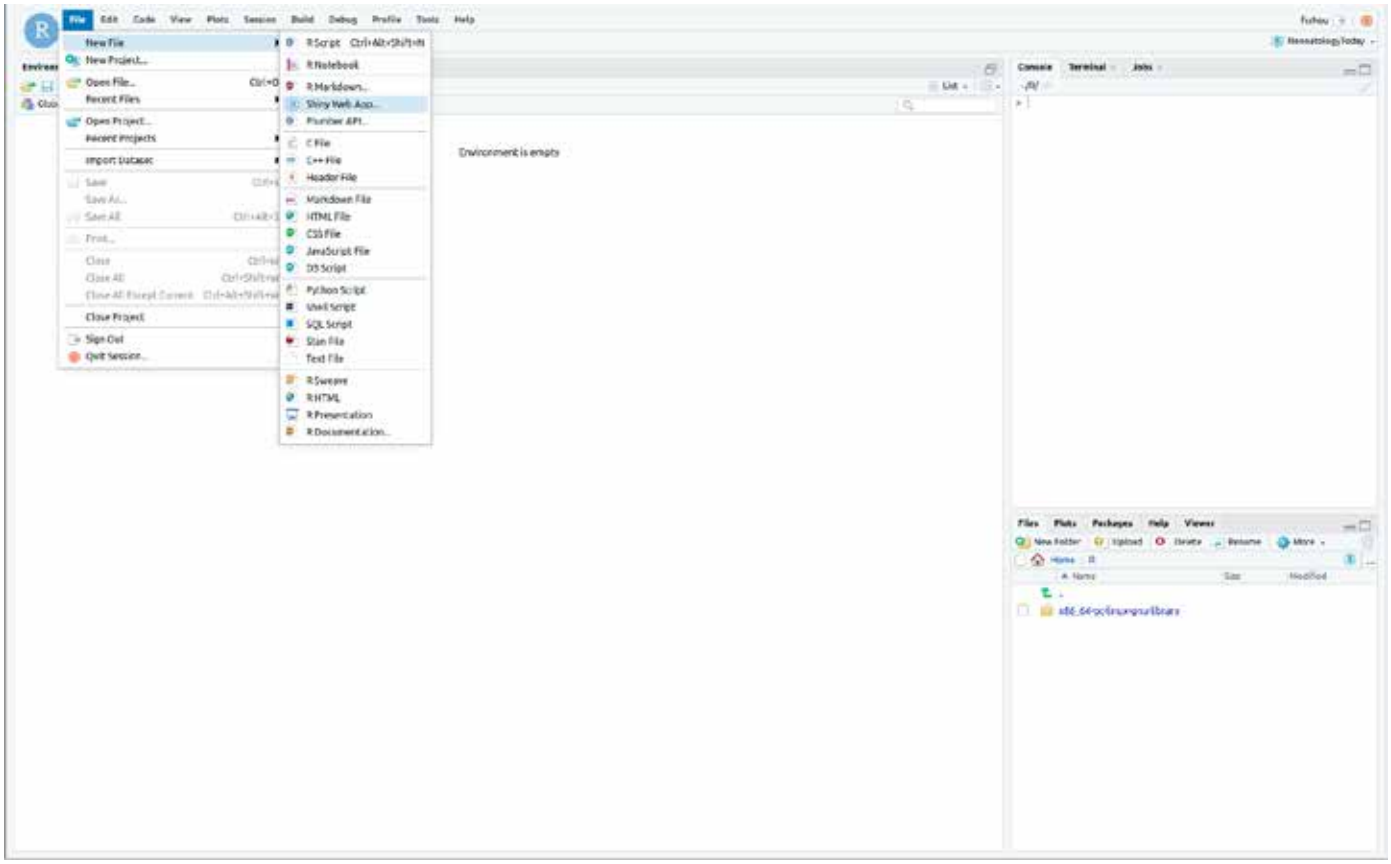


Figure 2

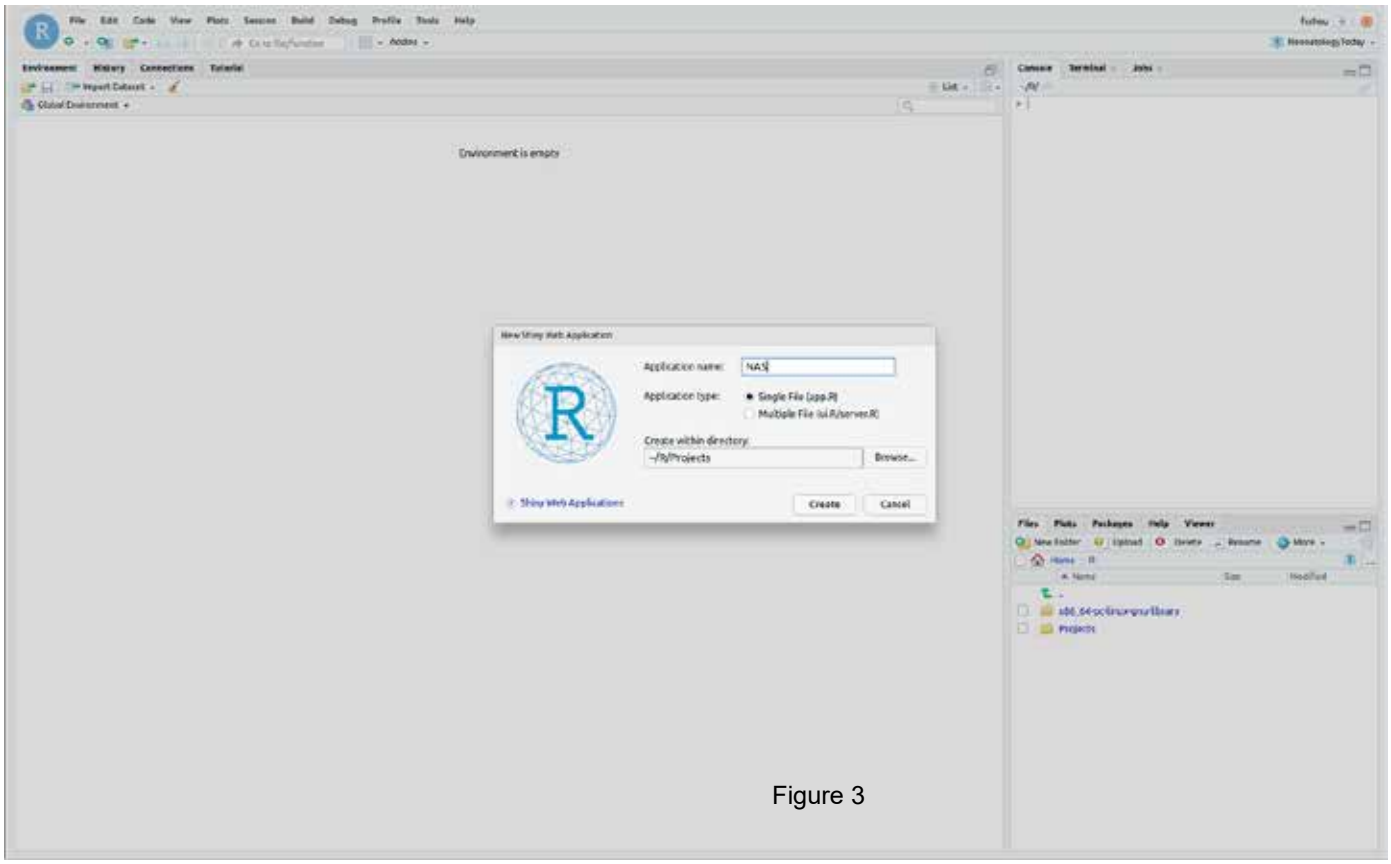


Figure 3

figure 3

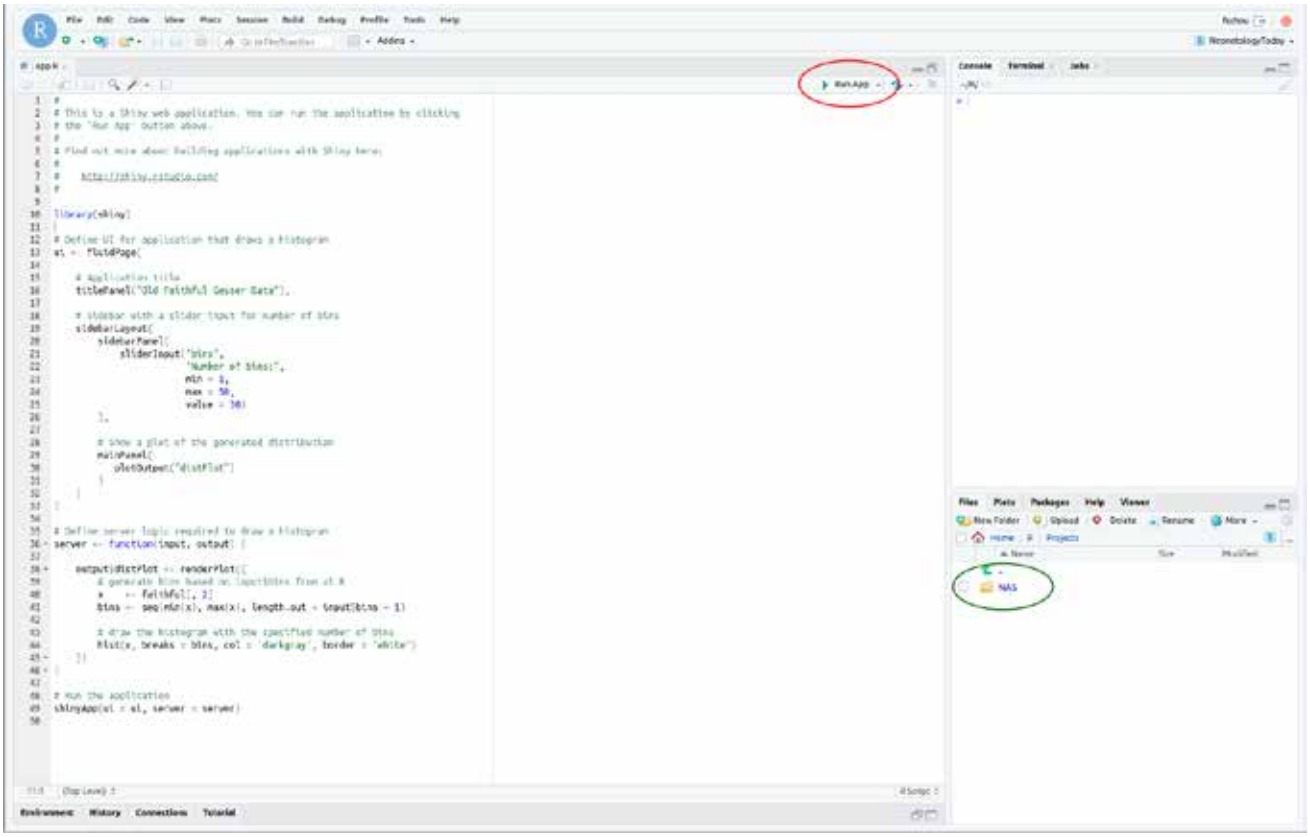
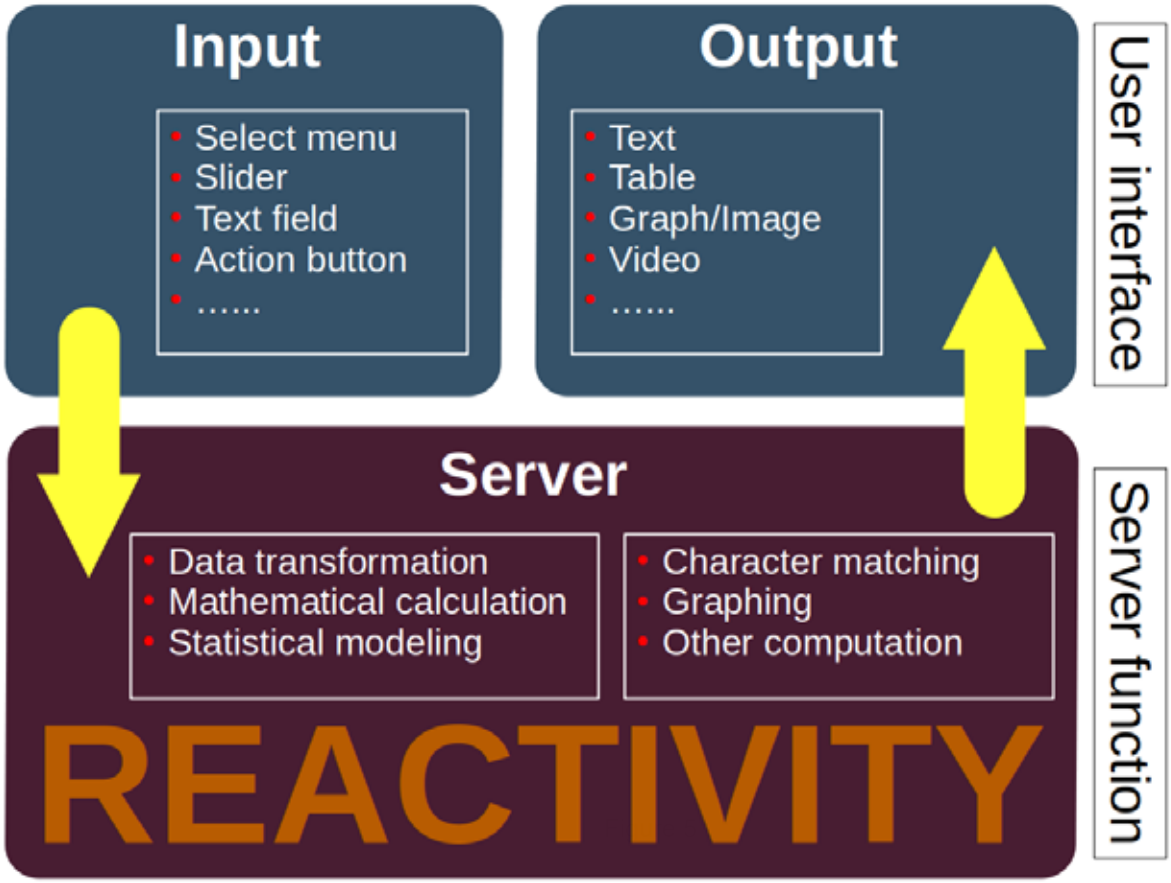
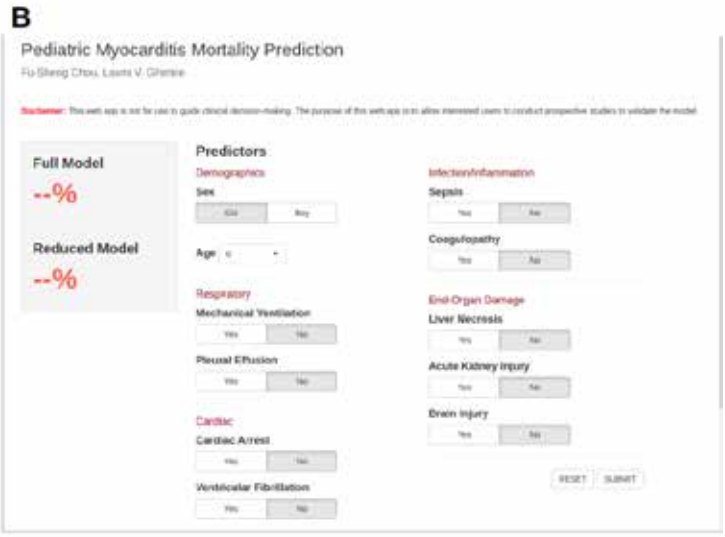


Figure 4





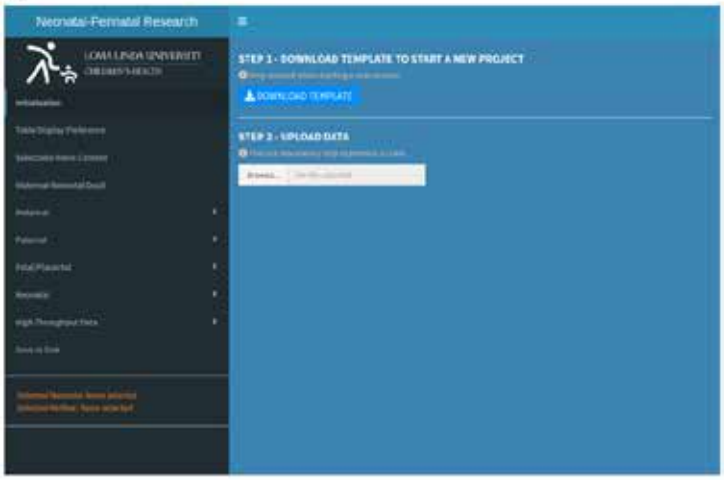


Table 1: Arguments for the siderInput() example.

Argument	Meaning
inputId = “bins”	a unique character id for R to refer to the widget
label = "Number of bins:"	the label of the widget displayed on the web page
min = 1	minimum number on the slider
max = 50	maximum number on the slider
value = 30	default value when the app is started

Table 2: Corresponding output and render functions.

output function	render function
plotOutput()	renderPlot()
tableOutput()	renderTable()
textOutput()	renderText()
imageOutput()	renderImage()
uiOutput()	renderUI()

References:

1. https://shiny.rstudio.com/reference/shiny/latest/sidebarLayout.html
2. https://rstudio.github.io/shinydashboard/get_started.html
3. https://shiny.rstudio.com/gallery/widget-gallery.html

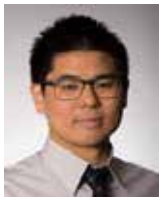
.

Acknowledgment: *The author would like to thank Dr. Shaina Lodhi in the Division of Neonatology, Department of Pediatrics at Loma Linda University School of Medicine for her insightful comments and constructive feedback on this article's content.*

Disclosure: The author identifies no conflict of interest

NT

Corresponding Author



Fu-Sheng Chou, MD, PhD -
Senior Associate Editor,
Director, Digital Enterprise
Neonatology Today
Assistant Professor of Pediatrics
Division of Neonatology, Department of Pediatrics
Loma Linda University Children’s Hospital
FChou@llu.edu