



## 2. server.R

Instrucciones que constituyen los componentes R de tu app. Para escribir server.R:

- A** Provee server.R con el mínimo de código necesario, `shinyServer(function(input, output) {})`.
- B** Define los componentes en R para tu app entre las llaves `{}` después de `function(input, output)`.
- C** Guarda cada componente R destinados para tu interfaz (UI) como `output$<nombre componente>`.
- D** Crea cada componente de salida con una función `render*`.
- E** Dale a cada función `render*` el código R que el servidor necesita para construir el componente. El servidor notará valores reactivos que aparecen en el código y reconstruirá el componente cada vez que estos valores cambian.
- F** Has referencia a valores en "widgets" con `input$<nombre del widget>`.

## 4. Reactividad

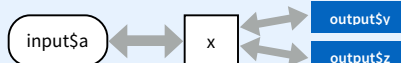
Cuando una entrada (input) cambia, el servidor reconstruye cada salida (output) que depende de ella (también si la dependencia es indirecta). Puedes controlar este comportamiento a través de la cadena de dependencias.

**render\*** - Una salida se actualiza automáticamente cuando una entrada en su función `render*` cambia.



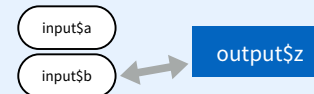
```
output$z <- renderText({
  input$a
})
```

**reactive** - usa `reactive` para crear objetos que se usaran en múltiples salidas.



```
x <- reactive({
  input$a
})
output$y <- renderText({
  x()
})
output$z <- renderText({
  x()
})
```

**isolate** - usa `isolate` para usar una entrada sin dependencia. Shiny no reconstruirá la salida cuando una entrada aislada cambia



```
output$z <- renderText({
  paste(
    isolate(input$a),
    input$b
  )
})
```

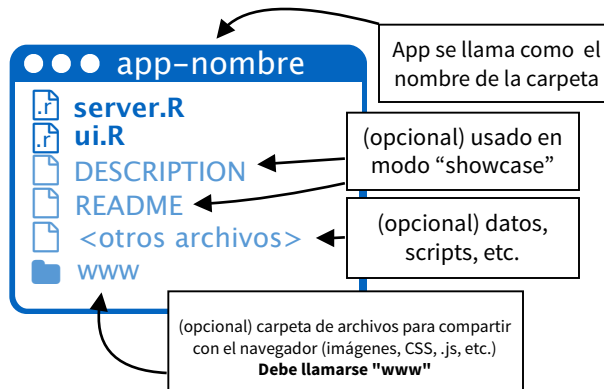
**observe** - usa `observe` para crear código que corre cuando una entrada cambia, pero que no crea un objeto de salida.



```
observe({
  input$a
  # código para correr
})
```

## 1. Estructura

Cada app es una carpeta que contiene un archivo `server.R` y comúnmente un archivo `ui.R` (opcionalmente contiene archivos extra)



## server.R

```
# carga paquetes, scripts, datos
A shinyServer(function(input, output) {E
  # crea variables específicos para usuario
  output$texto <- renderText({
    input$titulo
  })
  C output$gráfica <- D renderPlot({
    E x <- mtcars[ , input$x]F
    y <- mtcars[ , input$y]
    plot(x, y, pch = 16)
  })
})
```

## 3. Ejecución

Coloca código en el lugar donde correrá la menor cantidad de veces

- Corre una vez** - código puesto *fuera de shinyServer* solo corre una vez cuando inicias tu app. Úsalo para instrucciones generales. Crea una sola copia en memoria.
- Corre una vez por usuario** - código puesto *dentro de shinyServer* corre una vez por cada usuario que visita tu app (o refresca su navegador). Úsalo para instrucciones que necesitas dar por cada usuario del app. Crea una copia por cada usuario.
- Corre a menudo** - código puesto dentro de una función `render*`, `reactive`, o `observe` correrá muchas veces. Úsalo solo para código que el servidor necesita para reconstruir un componente UI después de que un widget cambia.

## funciones render\*

function	espera	crea
<code>renderDataTable</code>	objetos como tablas	tabla <code>DataTables.js</code>
<code>renderImage</code>	lista atributos imagenes	imagen HTML
<code>renderPlot</code>	gráfica	gráfica
<code>renderPrint</code>	salida impresa	texto
<code>renderTable</code>	objetos como tablas	tabla simple
<code>renderText</code>	cadena de caracteres	texto
<code>renderUI</code>	objeto "tag" o HTML	elemento UI (HTML)

**valores de entrada (input) son reactivos.**  
Deben estar rodeados por uno de:

- render\*** - crea un componente shiny UI (interfaz)
- reactive** - crea una expresión reactiva
- observe** - crea un observador reactivo
- isolate** - crea una copia no-reactiva de un objeto reactivo

# ui.R

## A shinyUI(fluidPage(

```

titlePanel("datos mtcars"),
B sidebarLayout(
  sidebarPanel(
    C textInput("titulo", "titulo gráfica:",
      value = "x v y"),

    selectInput("x", "Escoge una var x:"
      choices = names(mtcars),
      selected = "disp"),

    selectInput("y", "Escoge una var y:"
      choices = names(mtcars),
      selected = "mpg")
  ),

  mainPanel(
    h3(textOutput("texto")),
    plotOutput("plot")
  )
)
)

```

## C En cada panel o columna pon...



**Componentes R** - Los objetos de salida que has definido en **server.R**. Para colocar un componente:

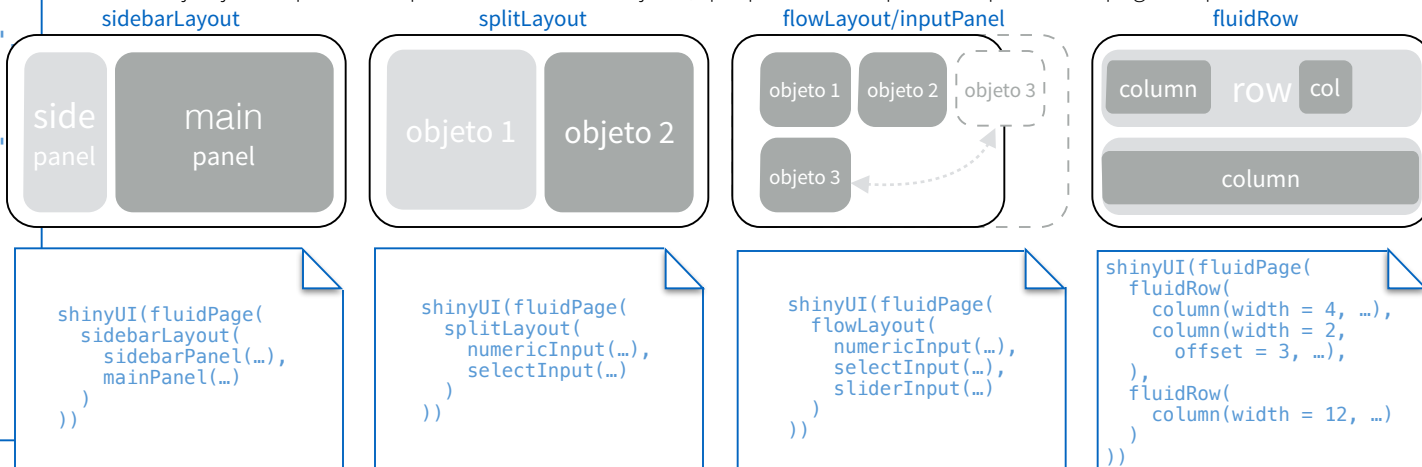
1. Selecciona la función **\*Output** que construye el tipo de objeto que quieres colocar en la UI.
2. Pasa la función **\*Output** a una cadena de caracteres correspondiente al nombre del objeto en **server.R**:  
`output$gráfica <- renderPlot({ ... })` ↔ `plotOutput("gráfica")`

### funciones \*Output

dataTableOutput	tableOutput
htmlOutput	textOutput
imageOutput	uiOutput
plotOutput	verbatimTextOutput

# 5. ui.R Una descripción de la interfaz (UI) de tu app, la página web que muestra tu app. Para escribir ui.R:

- A Incluye el mínimo de código necesario para ui.R, `shinyUI(fluidPage())`  
 \* nota: usa `navbarPage` en vez de `fluidPage` si quieres que tu app tenga múltiples páginas conectados con un navbar
- B Construye el plano para tu UI. `sidebarLayout` da una composición estándar cuando se usa con `sidebarPanel` y `mainPanel`. `splitLayout`, `flowLayout`, e `inputLayout` dividen la página en regiones equidistantes. `fluidRow` y `column` trabajan juntos para crear planos basados en rejillas, que puedes usar para componer una página o panel.



**Widgets** - El primer argumento de cada función de *widget* es el **<nombre>** del widget. Puedes acceder a su valor actual en **server.R** con `input$<nombre>`

widget	función	argumentos comunes
Botón de acción	actionButton	inputId, label
casilla	checkboxInput	inputId, label, value
grupo de casillas	checkboxGroupInput	inputId, label, choices, selected
selección de fechas	dateInput	inputId, label, value, min, max, format
selección rango fechas	dateRangeInput	inputId, label, start, end, min, max, format
subir archivo	fileInput	inputId, label, multiple
campo numerico	numericInput	inputId, label, value, min, max, step
botón de selección	radioButtons	inputId, label, choices, selected
casilla de selección	selectInput	inputId, label, choices, selected, multiple
deslizador	sliderInput	inputId, label, min, max, value, step
botón de envío	submitButton	text
campo de texto	textInput	inputId, label, value



**Elementos HTML** - Añade elementos html con funciones shiny similares a etiquetas HTML comunes.

a	tags\$col	tags\$form	tags\$input	tags\$output	tags\$sub
tags\$abbr	tags\$colgroup	h1	tags\$ins	tags\$param	tags\$summary
tags\$address	tags\$command	h2	tags\$kbd	tags\$pre	tags\$sup
tags\$area	tags\$data	h3	tags\$keygen	tags\$progress	tags\$table
tags\$article	tags\$datalist	h4	tags\$label	tags\$script	tags <tbody></tbody>
tags\$aside	tags\$dd	h5	tags\$legend	tags\$span	tags\$td
tags\$audio	tags\$del	h6	tags\$li	tags\$tbody	tags\$thead
tags\$b	tags\$details	tags\$thead	tags\$link	tags\$tr	tags\$tfoot
tags\$bbase	tags\$dfn	tags\$header	tags\$mark	tags\$tr	tags\$timestr
tags\$bdo	tags\$div	tags\$hgroup	tags\$map	tags\$tr	tags\$thead
tags\$blockquote	tags\$ddiv	hr	tags\$menu	tags\$tr	tags\$thead
tags\$body	tags\$em	HTML	tags\$meta	tags\$tr	tags\$thead
br	tags\$embed	tags\$i	tags\$meter	tags\$tr	tags\$thead
tags\$button	tags\$eventsource	tags\$iframe	tags\$nav	tags\$tr	tags\$thead
tags\$canvas	tags\$fieldset	tags\$img	tags\$noscript	tags\$tr	tags\$thead
tags\$caption	tags\$figcaption	includeCSS	tags\$object	tags\$tr	tags\$thead
tags\$cite	tags\$figure	tags\$includeMarkdo	tags\$span	tags\$tr	tags\$thead
code	tags\$figure	wn	tags\$strong	tags\$tr	tags\$thead
	tags\$footer	includeScript	tags\$style	tags\$tr	tags\$thead

## 6. Corre tu app

- runApp** - corre archivos locales
- runGitHub** - corre archivos alojados en [www.GitHub.com](http://www.GitHub.com)
- runGist** - corre archivos guardados como gist ([gist.github.com](http://gist.github.com))
- runURL** - corre archivos guardado en algún URL

## 7. Comparte tu app Presenta tu app como una página web accesible en línea

### ShinyApps.io

Aloja tus apps en el servidor de RStudio. Opciones gratis y pagas.  
[www.shinyapps.io](http://www.shinyapps.io)

### Shiny Server

Construye to propio servidor linux para alojar apps. Gratis y de código abierto.  
[shiny.rstudio.com/deploy](http://shiny.rstudio.com/deploy)

### Shiny Server Pro

Construye un servidor comercial con autenticación, gestión de recursos y más.  
[shiny.rstudio.com/deploy](http://shiny.rstudio.com/deploy)

