How to create applications using Shiny

ISGlobal

Barcelona, February 13-14th 2018

Part V: Advanced issues

Outline

Part V: Advanced issues

observe and observeEvent functions

observeEvent

- The **observeEvent** function is meant to execute instructions inside the Server section when **one** element is changed/updated.
- All the code inside `observeEvent` will be exectued only if this element is changed.

```
server <- function(input, output){
  ...
  observeEvent(input$element,{
    ...
    instructions
    ...
  })
  ...
}
```

Example: Add a row in a stored data frame.

```r
# initiate with an empty data.frame
# these first two lines must be executed just once!!
data <- data.frame(name = character(), age = numeric())
save(data, file="data.rda")

# the app begins here
ui <- fluidPage(
  textInput("name", "Name", ""),
  numericInput("age", "Age", NA),
  actionButton("add", "Add")
)

server <- function(input, output){
  observeEvent(input$add,{
    load("data.rda")
    newrow <- data.frame(name=input$name, age=input$age)
    data <- rbind(data, newrow)
    save(data, file="data.rda")
  })
}

shinyApp(ui, server)
```

Check how in “data.rda” new rows are being added.

observe

- The first argument of `observeEvent` function is a single element.
- But, what happens if you desire to execute the instructions if one out of **several** elements change?
- Then **observe** function is used instead of `observeEvent`. You can use `isolate`.

```r
server <- function(input, output){
  ...
  observe({
    input$element1
    input$element2
    isolate({
      instructions
    })
  })
  ...
}
```

Example

The code is executed when “age” or “chol” inputs elements are udpated.

```
library(shinyFeedback)

ui <- fluidPage(
  useShinyFeedback(),

  numericInput("age", "Age", NA),
  numericInput("chol", "Cholesterol", NA)

)

server <- function(input, output) {

  observe({
   feedback("age", condition=input$age<30,
      text="Age must be > 30", color="red")
   feedback("chol", condition=input$chol>1000,
      text="Cholesterol must be < 1000", color="red")
  })

}

shinyApp(ui, server)
```

Updating elements

Updating elements

- Unlike `uiOUtput / renderUI`, using the **`update***`** functions only those specified arguments are modified and the others remain as they are.
- **`update***`** functions are used inside Server section.
- Specifically, they are called inside `observe` or `observeEvent`.
- Note the argument **`session`** when defining Server function.

Initialization	Update	Modifiable arguments
`textInput`	`updateTextInput`	`label`, `value`
`numericInput`	`updateNumericInput`	`label`, `value`
`checkboxInput`	`updateCheckboxInput`	`label`, `value`
`radioButtons`	`updateRadioButtons`	`label`, `choices`, `selected`, `inline`
`selectInput`	`updateSelectInput`	`label`, `choices`, `selected`
`sliderInput`	`updateSliderInput`	`label`, `value`, `min`, `max`, `step`
`actionButton`	`updateActionButton`	`label`, `icon`
`bsButton {shinyBS}`	`updateButton`	`label`, `icon`, `style`, `disabled`

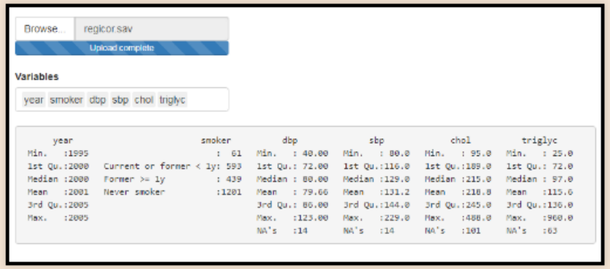
Example: Add a row in a stored data frame

Let's recover a previous example but now input elements are re-seted once “add” button is pressed.

```r
data <- data.frame(name = character(), age = numeric())
save(data, file="data.rda")

ui <- fluidPage(
  textInput("name", "Name", ""),
  numericInput("age", "Age", NA),
  actionButton("add", "Add")
)

server <- function(input, output, session){
  observeEvent(input$add,{
    load("data.rda")
    newrow <- data.frame(name=input$name, age=input$age)
    data <- rbind(data, newrow)
    save(data, file="data.rda")
    # re-set input elements to blank
    updateTextInput(session, "name", value="")
    updateNumericInput(session, "age", value=NA)
  })
}

shinyApp(ui, server)
```

Example: variable list

```r
library(Hmisc)

ui <- fluidPage(
  fileInput("file", ""),

  selectInput("vars", "Variables", choices=NULL,
              multiple=TRUE),

  verbatimTextOutput("summary")
)

server <- function(input, output, session){
  dat <- reactive({
    if (is.null(input$file))
      return(invisible(NULL))
    spss.get(input$file$datapath)
  })
  output$summary <- renderPrint({
    summary(dat()[,input$vars])
  })

  observe({
    updateSelectInput(session, "vars",
                      choices = names(dat()))
  })

}

shinyApp(ui, server)
```

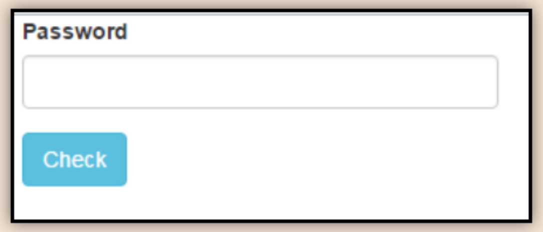


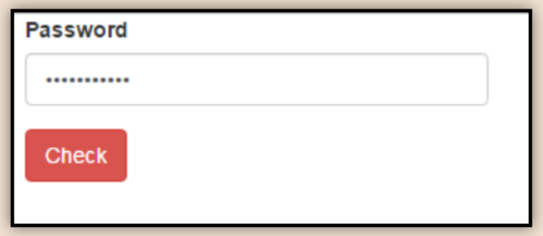
```
      year                           smoker         dbp              sbp              chol            triglyc
 Min.   :1995                           :  61   Min.   : 40.00   Min.   : 80.0   Min.   : 95.0   Min.   : 25.0
 1st Qu.:2000   Current or former < 1y: 593   1st Qu.: 72.00   1st Qu.:116.0   1st Qu.:189.0   1st Qu.: 72.0
 Median :2000   Former >= 1y          : 439   Median : 80.00   Median :129.0   Median :215.0   Median : 97.0
 Mean   :2001   Never smoker          :1201   Mean   : 79.66   Mean   :131.2   Mean   :218.8   Mean   :115.6
 3rd Qu.:2005                                 3rd Qu.: 86.00   3rd Qu.:144.0   3rd Qu.:245.0   3rd Qu.:136.0
 Max.   :2005                                 Max.   :123.00   Max.   :229.0   Max.   :488.0   Max.   :960.0
                                              NA's   :14       NA's   :14      NA's   :101     NA's   :63
```

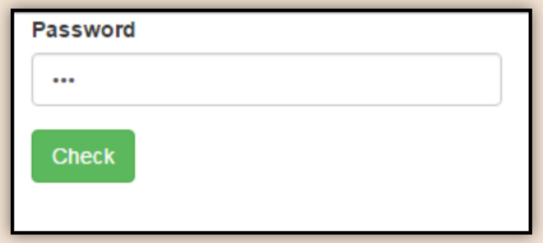
Exercise: use renderUI and uiOutput

Example 2: Button (style)

```r
library(shinyBS)

ui <- fluidPage(
  passwordInput("pass", "Password"),
  bsButton("check", "Check", style="info")
)

server <- function(input, output, session){

  observeEvent(input$check, {
    if (input$pass=='')
      updateButton(session, "check", style="warning")
    else {
      if (input$pass=='123')
        updateButton(session, "check", style="success")
      else
        updateButton(session, "check", style="danger")
    }
  })

}

shinyApp(ui, server)
```

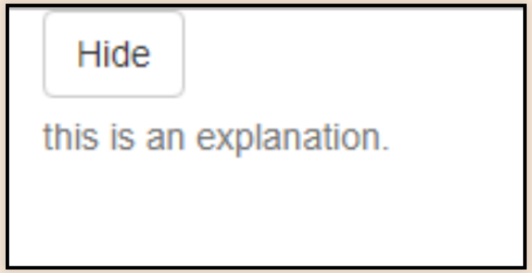




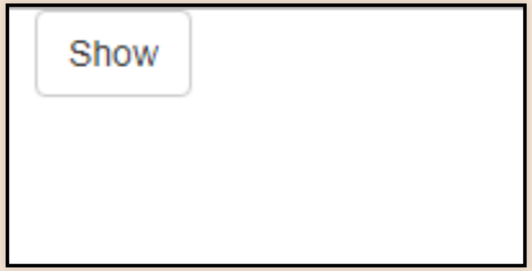


Example 2: Button (label)

```r
ui <- fluidPage(
  actionButton("help", "Hide", width = "60px"),
  conditionalPanel(
    condition = "input.help%2==0",
    helpText("this is an explanation.")
  )
)

server <- function(input, output, session){

  observeEvent(input$help, {
    if (input$help%%2==0)
      updateActionButton(session, "help", label="Hide")
    else {
      updateActionButton(session, "help", label="Show")
    }
  })

}

shinyApp(ui, server)
```



this is an explanation.



Reactive variables

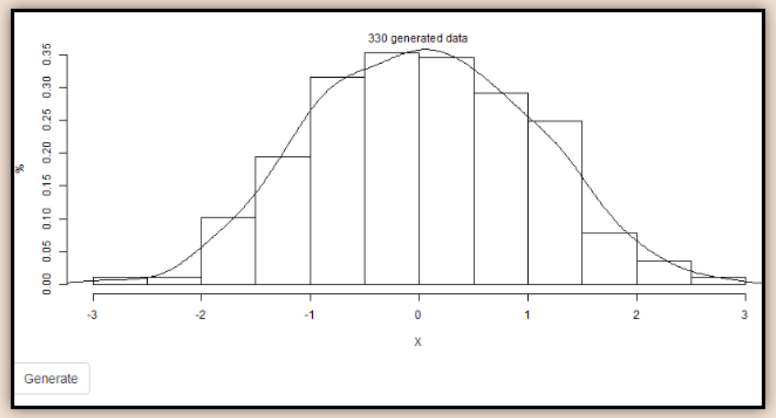
Reactive variables

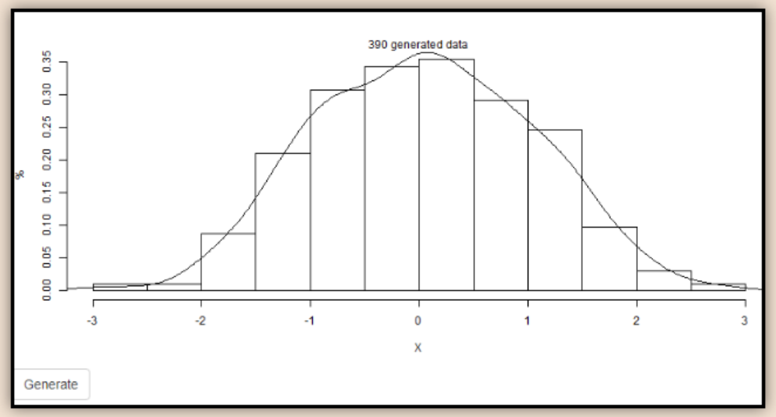
- **Reactive variables** are objects whose values are modified in a "reactive" way
- They exists inside the server section.
- Every *reactive values* type object is an element of a list which is iniciated using `reactiveValues` function.

```r
server <- function(input, output){

  rv <- reactiveValues()
  rv$element <- 0
  ...

}
```

Example 1: Cumulating data plot

- Generate data from a normal distribution when pressing a button.
- The generated data must be added to already generated ones.
- Plot an histogram with all the cumulated data.



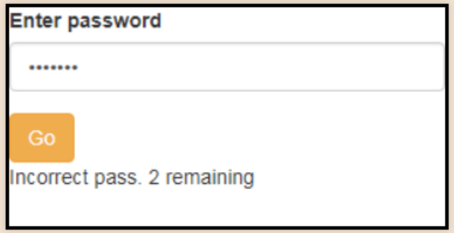


```r
ui<-bootstrapPage(
  plotOutput("plot"),
  actionButton("go", "Generate")
)

server<-function(input,output){

  rv <- reactiveValues()
  rv$numbers <- numeric()

  observeEvent(input$go,{
    rv$numbers <- c(rv$numbers,rnorm(10))
  })

  output$plot <- renderPlot({
    if (length(rv$numbers)==0) return(invisible(NULL)) # no data yet
    hist(rv$numbers, freq=FALSE, xlab="X", ylab="%",main="")
    lines(density(rv$numbers))
    mtext(paste(length(rv$numbers), "generated data"))
  })
}

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

Note that `renderPlot` is executed when `rv$numbers` change.

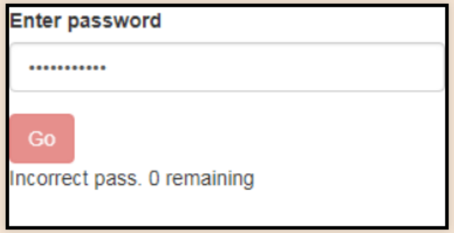
Example 2: Enter a password

- Once the password is typed, check it by pressing a button.
- When 3 attempts are achieved without success, the button must be disabled and coloured in red.
- If the password is correct the button must be disabled and coloured in green.
- The number of attempts will be stored in a reactive element (`reactiveValues`).


Enter password

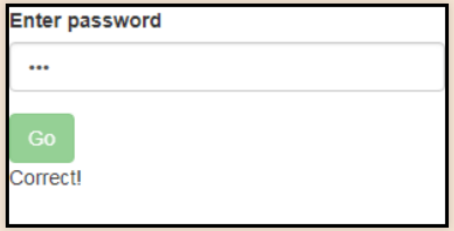
•••••••

Incorrect pass. 2 remaining


Enter password

•••••••••••

Incorrect pass. 0 remaining


Enter password

•••

Correct!

```r
library(shinyBS)
pass<-"123"

ui<-bootstrapPage(
  passwordInput("idpass","Enter password",""),
  bsButton("idbutton", "Go"),
  uiOutput("result")
)

server<-function(input,output,session){
  # iniciate number of attempts to 3
  rv <- reactiveValues(attempts=3)
  # do something when check buttonis pressed
  observeEvent(input$idbutton,{
    if (input$idpass == "") # no attempts yet
      updateButton(session,"idbutton",style="info")
    if (input$idpass!="" & input$idpass!=pass){ # incorrect password
      rv$attempts<<-rv$attempts-1
      if (rv$attempts == 0){ # no attempts remaining
        updateButton(session,"idbutton",style="danger",disabled=TRUE)
      } else { # only 1 remaining attempt
        updateButton(session,"idbutton",style="warning",disabled=FALSE)
      }
    }
    if (input$idpass==pass){
      updateButton(session,"idbutton",style="success",disabled=TRUE)
    }
  })

  output$result<-renderUI({
    if (input$idbutton==0) return(invisible(NULL))
    isolate({
      if (input$idpass!="" & input$idpass!=pass)
        return(paste("\n\nIncorrect pass. ",rv$attempts,"remaining"))
      if (input$idpass=="")
        return("\n\nEnter pass")
      if (input$idpass==pass){
        return("\n\nCorrect!")
      }
    })
  })
}

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

Toggle, show, hide & disable

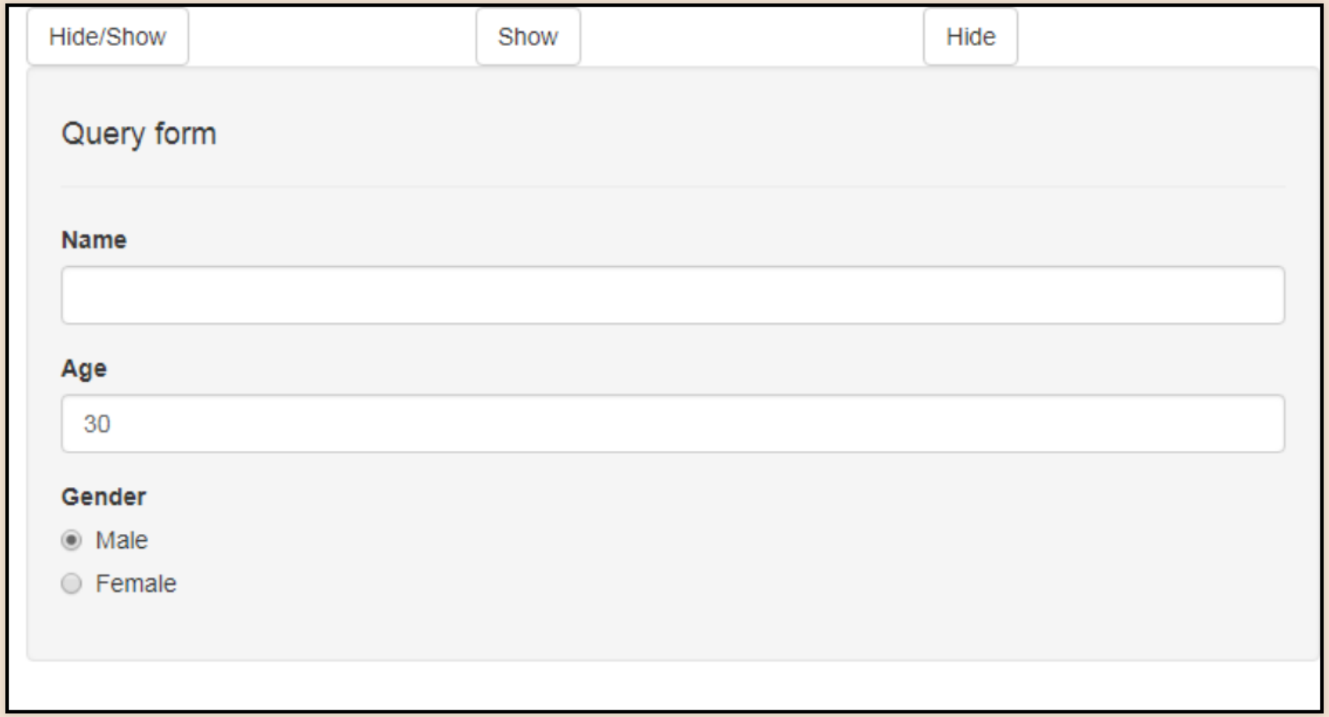
Toggle, show, hide & disable

- The **shinyjs** package, among other features, allows:
  - Hide/Show form widgets (`hide`, `show`, `toggle`)
  - Enable or disable buttons or other input widgets (`disable`)
- It is available on CRAN:

```
install.packages(shinyjs)
```

- For more info, visit its website.

Example 1. Buttons (hide, show, toggle)



Query form

Name

Age

30

Gender

(●) Male

( ) Female

UI

```
ui <- fluidPage(

  useShinyjs(), # Set up shinyjs

  fluidRow(
    column(4, actionButton("btntoggle",
                           "Hide/Show")),
    column(4, actionButton("btnshow", "Show")),
    column(4, actionButton("btnhide", "Hide"))
  ),

  hidden(
    wellPanel(id="elem",
      h4("Query form"),
      hr(),
      textInput("name", "Name", ""),
      numericInput("age", "Age", 30),
      radioButtons("gender", "Gender",
                   c("Male", "Female"))
    )
  )
)
```

Server

```
server <- function(input, output) {

  observeEvent(input$btntoggle, {
    shinyjs::toggle("elem", TRUE, "fade")
  })

  observeEvent(input$btnshow, {
    shinyjs::show("elem", TRUE, "slide")
  })

  observeEvent(input$btnhide, {
    shinyjs::hide("elem", FALSE)
  })

}
```

Example 2. Password

1. Make the app visible only if the password is correct.
2. Once the correct password is introduced <u>the password input widget and the check button must disapear</u>

```
ui <- fluidPage(

  useShinyjs(),

  div(id="passScreen",
    passwordInput("pass","Password",""),
    actionButton("check","Check")
  ),

  shinyjs::hidden(
    div(id="myapp",
      titlePanel("Hello Shiny!"),
      sidebarLayout(
        sidebarPanel(
          sliderInput("obs","Number obs.",
              min=1,max=1000,value=500)
        ),
        mainPanel(
          plotOutput("distPlot")
        )
      )
    )
  )
)
```

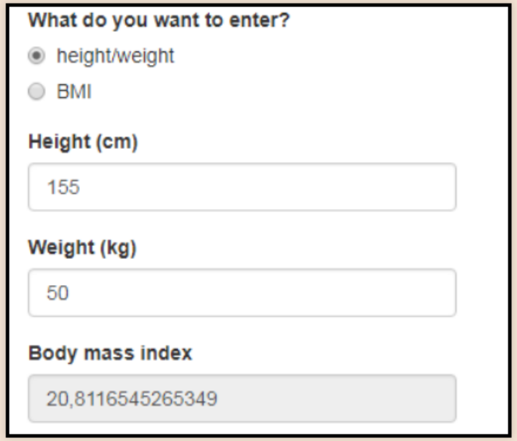
```
server <- function(input, output) {

  observeEvent(input$check, {
    if (input$pass=='123'){
      shinyjs::show("myapp", FALSE)
      shinyjs::hide("passScreen", FALSE)
    } else {
      shinyjs::hide("myapp", FALSE)
      shinyjs::show("passScreen", FALSE)
    }
  })

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

}
```

Try it here

Example 3. Body mass index

- You can enter either weight and height or body mass index (bmi).
- If user enter height and weight, bmi input widgets must be disabled but visible and updated according to bmi formula (weight/height^2)
- If user enter bmi, height and weight input widgets must be hidden.

```r
library(shinyjs)

ui <- fluidPage(
  useShinyjs(),
  radioButtons("what","What do you want to enter?",
               c("height/weight"=1, "BMI"=2)),
  numericInput("height","Height (cm)",NA),
  numericInput("weight","Weight (kg)",NA),
  numericInput("bmi","Body mass index",NA)
)

server <- function(input, output, session) {
  observe({
    if (input$what==1){
      shinyjs::disable("bmi", FALSE)
      updateNumericInput(session, "bmi",
          value=input$weight/(input$height/100)^2)
      shinyjs::show("height", FALSE)
      shinyjs::show("weight", FALSE)
    } else {
      shinyjs::enable("bmi", FALSE)
      shinyjs::hide("height", FALSE)
      shinyjs::hide("weight", FALSE)
    }
  })
}

shinyApp(ui, server)
```


What do you want to enter?

[✓ height/weight]

BMI

Height (cm)

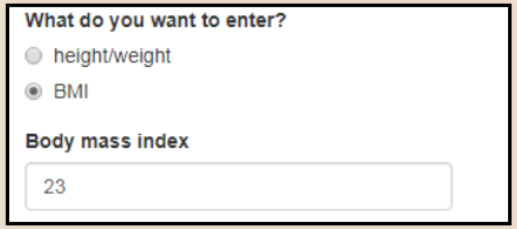
155

Weight (kg)

50

Body mass index

20,8116545265349


What do you want to enter?

height/weight

[✓ BMI]

Body mass index

23

Exercise

Exercise

From the app created in part IV:

- Add a password that be hidden once the pass is correct (123)

Try it here