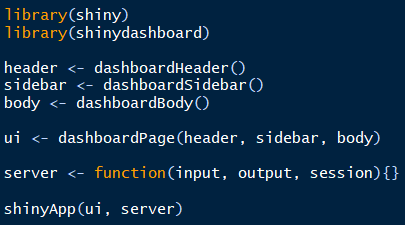
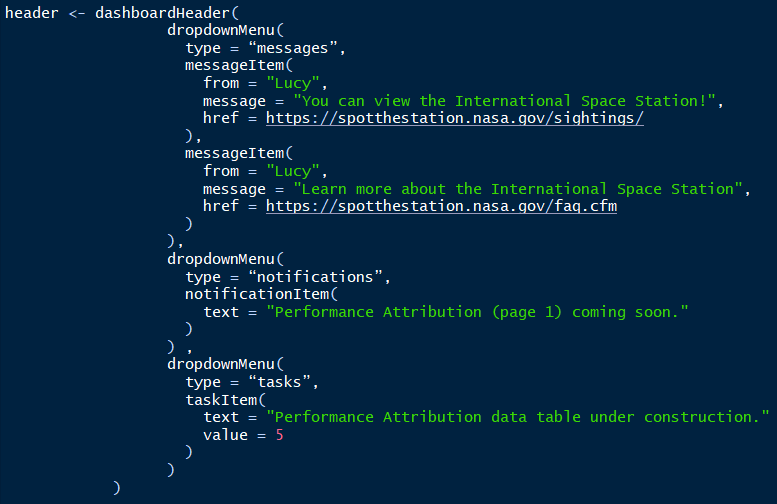
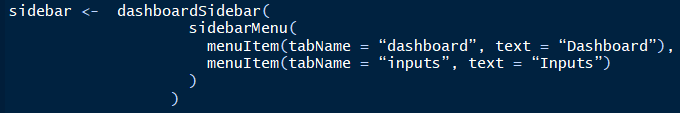
Building Dashboards with shinydashboard

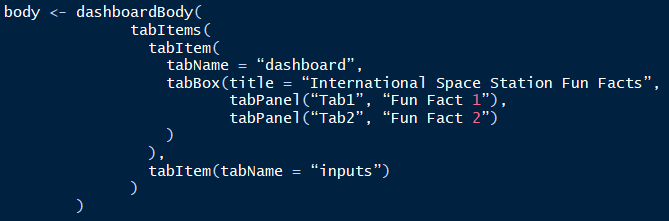
**Building Static Dashboards**

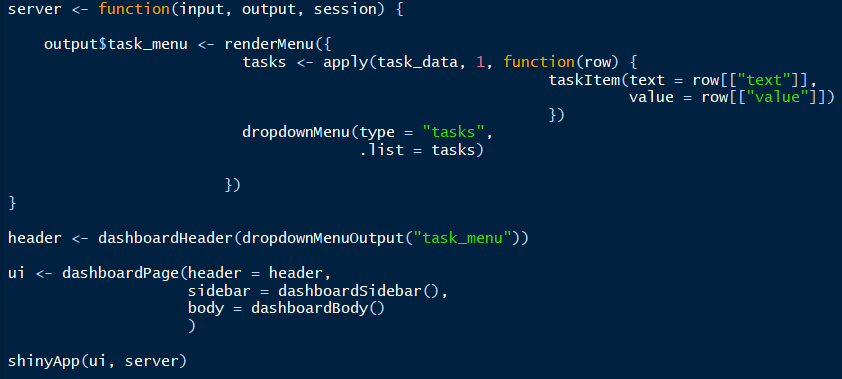


***Basic anatomy of a shiny dashboard***







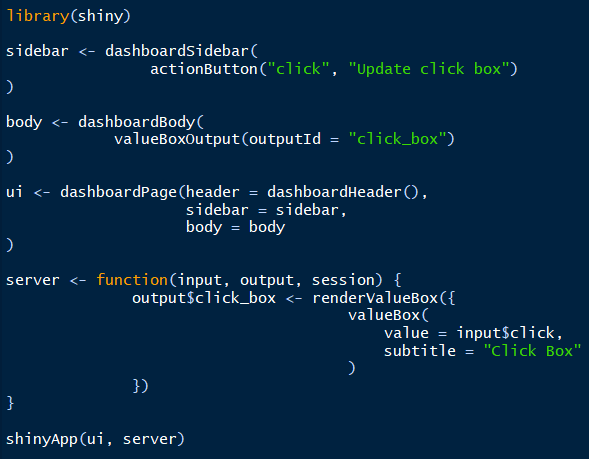
The snippets shown on the previous page display the components of a basic shinydashboard. Notice that the message dropdowns in the snippets are static. You can make the message dynamic by connecting the information to a dataframe as shown in the snippet below.

In the above snippet, information is stored in the task\_data dataframe. Updating the task\_data causes the content of the dropdownMenuOutput to update dynamically.

In addition to the renderMenu / dropdownMenuOutput pair, there are two other pairs that are useful:

renderInfoBox() 🡨🡪 infoBoxOutput()

renderValueBox() 🡨🡪 valueBoxOutput()



**Building Dynamic Dashboards**

selectInput(

inputId = …,

label = “…”,

choices = c(…)

)

Types of input widgets (ui)

* actionButton()
* checkboxInput()
* dateInput()
* numericInput()
* radioButtons()
* selectInput()
* sliderInput()
* textInput()

Types of render functions (server) Corresponding output functions (ui)

* renderPrint() verbatimTextOutput(), textOutput()
* renderText() textOutput()
* renderTable() tableOutput()
* renderDataTable() dataTableOutput()
* renderPlot() plotOutput()
* renderImage() imageOutput()
* renderUI() uiOutput(), htmlOutput()

textOutput(outputId = “printed\_object”)

output$printed\_object <- renderPrint({

“print me”

})

output$printed\_object <- renderPrint({

“print me”

})

HTML or Shiny tag object

**Building Dynamic Dashboards (continued)**

Using real-time data with Shiny dashboards using reactiveFileReader() or reactivePoll():

filePath <- starwars\_url

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

reactive\_data <- reactiveFileReader(

intervalMillis = 1000,

session = session,

filePath = filePath,

readFunc = function(filePath) {

read.csv(url(filePath))

}

)

output$print\_data <- renderPrint({

reactive\_data()

})

}

**Optimizing Performance**

There are various ways of optimizing code. Placement of code, primarily dealing with data loading, is critical to app optimization. For example:

* Static data placed outside the server function will be executed once when the app is launched.
* Static data placed inside the server function but not within a reactive function, will be loaded each time a new user visits your app.
* Static data placed inside the server function and within a reactive function will be reloaded each time a user invalidates an input.

<https://www.rstudio.com/resources/videos/profiling-and-performance/> is an additional source for learning more tips and tricks related to performance optimization.