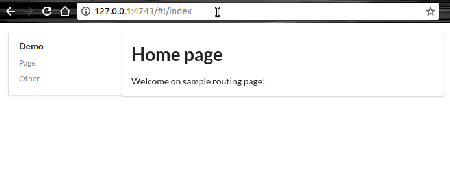
In a web application, routing is the process of using URLs to drive the user interface. Routing adds more possibilities and flexibility while building a complex and advanced web application, offering dividing app into separated sections.



**New features**

Now it’s time for shiny.router. Our shiny.router package provides you an easy solution how to add routing to your Shiny application.

Separated server for each bookmark

Now each bookmark can be isolated and fully working shiny app. The new feature allows you not only to separate UI for each bookmark – you may also define its own server now. Just check below example!

library(shiny)

library(shiny.router)

# This creates UI for each page.

page <- function(title, content) {

div(

titlePanel(title),

p(content),

uiOutput("power\_of\_input")

)

}

# Part of both sample pages.

home\_page <- page("Home page", "This is the home page!")

side\_page <- page("Side page", "This is the side page!")

# Callbacks on the server side for the sample pages

home\_server <- function(input, output, session) {

output$power\_of\_input <- renderUI({

HTML(paste(

"I display **square** of input and pass result to output$power\_of\_input: ",

as.numeric(input$int) ^ 2))

})

}

side\_server <- function(input, output, session) {

output$power\_of\_input <- renderUI({

HTML(paste(

"I display **cube** of input and **also** pass result to output$power\_of\_input: ",

as.numeric(input$int) ^ 3))

})

}

# Create routing. We provide routing path, a UI as well as a server-side callback for each page.

router <- make\_router(

route("home", home\_page, home\_server),

route("side", side\_page, side\_server)

)

# Create output for our router in main UI of Shiny app.

ui <- shinyUI(fluidPage(

shiny::sliderInput("int", "Choose integer:", -10, 10, 1, 1),

router\_ui()

))

# Plug router into Shiny server.

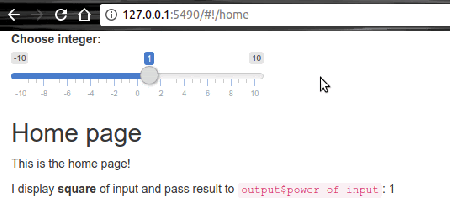
server <- shinyServer(function(input, output, session) {

router(input, output, session)

})

# Run server in a standard way.

shinyApp(ui, server)



**Pass parameters to an app using GET URL variables**

library(shiny)

library(shiny.router)

# Main page UI.

home\_page <- div(

titlePanel("Home page"),

p("This is the home page!"),

uiOutput("power\_of\_input")

)

# Creates routing. We provide routing path, a UI as well as a server-side callback for each page.

router <- make\_router(

route("/", home\_page, NA)

)

# Create output for our router in main UI of Shiny app.

ui <- shinyUI(fluidPage(

shiny::sliderInput("int", "Choose integer:", -10, 10, 1, 1),

router\_ui()

))

# Plug router into Shiny server.

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

router(input, output, session)

component <- reactive({

if (is.null(get\_query\_param()$add)) {

return(0)

}

as.numeric(get\_query\_param()$add)

})

output$power\_of\_input <- renderUI({

HTML(paste(

"I display input increased by add GET parameter from app url and pass result to output$power\_of\_input: ",

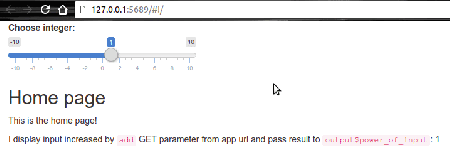
as.numeric(input$int) + component()))

})

})

# Run server in a standard way.

shinyApp(ui, server)



**Operate routing from the server side**

* route\_link – function for changing url for bookmark by adding hashbang (#!) prefix,
* change\_page – function for changing the currently displayed page,
* get\_page – function to extract “hash” part of the url,
* is\_page – function that verifies if current page is was passed succesfuly.

library(shiny)

library(shiny.router)

# This generates menu in user interface with links.

menu <- (

tags$ul(

tags$li(a(class = "item", href = route\_link("home"), "Home page")),

tags$li(a(class = "item", href = route\_link("side"), "Side page"))

)

)

# This creates UI for each page.

page <- function(title, content) {

div(

menu,

titlePanel(title),

p(content),

actionButton("switch\_page", "Click to switch page!")

)

}

# Both sample pages.

home\_page <- page("Home page", uiOutput("current\_page"))

side\_page <- page("Side page", uiOutput("current\_page"))

# Creates router. We provide routing path, a UI as

# well as a server-side callback for each page.

router <- make\_router(

route("home", home\_page, NA),

route("side", side\_page, NA)

)

# Create output for our router in main UI of Shiny app.

ui <- shinyUI(fluidPage(

router\_ui()

))

# Plug router into Shiny server.

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

router(input, output, session)

output$current\_page <- renderText({

page <- get\_page(session)

sprintf("Welcome on %s page!", page)

})

observeEvent(input$switch\_page, {

if (is\_page("home")) {

change\_page("side")

} else if (is\_page("side")) {

change\_page("home")

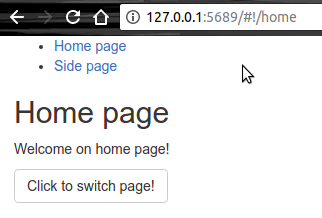
}

})

})

# Run server in a standard way.

shinyApp(ui, server)



Styling – Bootstrap and Semantic UI

You can suppress [Bootstrap](https://getbootstrap.com/) dependency on the specified bookmark. You can switch between Bootstrap and Semantic UI pages or disable styles. This is especially useful when using both Bootstrap and [semantic-UI](https://semantic-ui.com/) frameworks in one application.

library(shiny)

library(shiny.router)

library(shiny.semantic)

# Both sample pages.

bootstrap\_page <- fluidPage(

sidebarLayout(

sidebarPanel(

sliderInput("obs\_bootstrap",

NULL,

min = 0,

max = 100,

value = 50,

step = 1)

),

mainPanel(

p("Selected value:"),

textOutput("value\_bootstrap")

)

)

)

semanticui\_page <- semanticPage(

slider\_input("obs\_semantic",

min = 0,

max = 100,

value = 50,

step = 1),

p("Selected value:"),

textOutput("value\_semantic")

)

# Creates router. We provide routing path, a UI as

# well as a server-side callback for each page.

router <- make\_router(

route("bootstrap", bootstrap\_page),

route("semantic", semanticui\_page),

page\_404 = page404("You opened non existing bookmark!")

)

# Create output for our router in main UI of Shiny app.

ui <- shinyUI(

tagList(

tags$head(

singleton(disable\_bootstrap\_on\_bookmark("semantic"))

),

router\_ui()

)

)

# Plug router into Shiny server.

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

router(input, output, session)

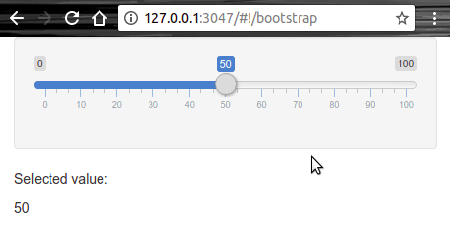
output$value\_bootstrap <- renderText(input$obs\_bootstrap)

output$value\_semantic <- renderText(input$obs\_semantic)

})

# Run server in a standard way.

shinyApp(ui, server)

****

We are planning to constantly work on the package to make it more versatile. As next steps we want to allow passing parameters between separated bookmarks servers and ability to save application state. We hope that you will appreciate improvements we did within last two years.