In this tutorial, I will go through the process of resembling the inertUI and removeUI

functions in Shiny with my own custom message handler in JavaScript and jQuery.

Let’s jump into it.

First, we will be creating a boot\_cards function with the help of Bootstrap that we will dynamically insert.

# Creating A Function That Creates the HTML Cards

boot\_cards <- function(title = NULL, body = NULL) { boot\_cards <-

div(

class = "class = col-md-3 my-cards", div(

class = "panel panel-default", div(

class = "panel-heading", p(title)

),

div(

class = "panel-body", p(body)

)

)

)

return(boot\_cards)

}

Then we are creating the UI part of the application.

# R Shiny Custom Message Handler UI Part

At the head of the document, we put the JavaScript and jQuery code that will dynamically remove and add cards. Then we specify a sidebarPanel() where we will be adding two textInputs(). One where we can specify the header and body of the card. Then, there are two action buttons. One button will delete the last card and the other one will insert a card after the placeholder id. We are also including a script that will tell us if the add button or remove button was clicked.

ui <- shiny::fluidPage(

shiny::singleton( tags$head(tags$script(src = "test.js"))

),

div(

class = "row", div(

shiny::sidebarPanel( id = "sidebar",

shiny::textInput(inputId = "header\_add", label = "Add Title", placeholder = "Title"),

shiny::textInput(inputId = "body\_add", label = "Add Body", placeholder = "Body"),

shiny::actionButton(inputId = "remove", label = "Remove Card"), shiny::actionButton(inputId = "add", label = "Add Card")

)

),

div(id = "placeholder")

),

shiny::includeScript("www/button\_click.js")

)

# R Shiny Custom Message Handler – From R to JavaScript

Let’s jump to the server-side. First, we will be creating a cards function. With this function, we will be communicating from R to JavaScript. We will be sending the Bootstrap cards and which button has been clicked (add or remove) to JavaScript. With the help of JavaScript and jQuery, the cards with the appropriate text will be inserted dynamically.

cards <- function(item, button, session = shiny:: getDefaultReactiveDomain()) {

session$sendCustomMessage( type = 'add-remove-cards', message = list(

card = as.character(item), add\_remove = button

)

)

}

The function above sends an R list which will be received by JavaScript as JSON. The list is sent to the custom handler we are creating.

$(function() {

Shiny.addCustomMessageHandler('add-remove-cards', function(message) {

var card = $.parseHTML(message.card); var btn = message.add\_remove

var card\_len = $(".my-cards").length;

if(btn === "add") { if(card\_len === 0) {

$(card).insertAfter($('#placeholder'));

} else {

$(card).insertAfter($('.my-cards:last'));

}

} else {

$(".my-cards").last().remove()

}

});

});

We can get the elements of the R list with message.card and message.add\_remove. If the add button was clicked and no card has been inserted yet, we will create the first card after the placeholder id. If a card already exists, we will be placing the HTML after the last existing card. If we want to remove cards, we are going to remove the last card. So certainly less flexible than removeUI and insertUI. However, one can adjust the function’s functionality. The custom message handler was added via Shiny.addCustomMessageHandler.

# From JavaScript to R – JavaScript method Shiny.onInputChange

Now, we will be discussing how to send a message from JavaScript to R. We can do that with the JavaScript method Shiny.onInputChange.

$("button").click(function() { var id = this.id;

Shiny.setInputValue("button\_clicked", id, {priority: "event"})

});

When a button is clicked, we get the id and will send it to R, which will send it then again to JavaScript. That is not very efficient and I should have included the click event in the previous function so that we do not have to send it from JavaScript to R and then back to JavaScript again. However, I left the code for educational purposes 

# R Shiny Server-Side

Because everything happens in the Browser with jQuery and JavaScript, the server-side is very simple.

server <- function(input, output, session) { shiny::observeEvent(input$remove | input$add, {

cards(

item = boot\_cards( input$header\_add, input$body\_add

),

button = input$button\_clicked

)

})

}

When either action button is clicked we send the information about what button is clicked from R to JavaScript where we then add or delete the cards dynamically.