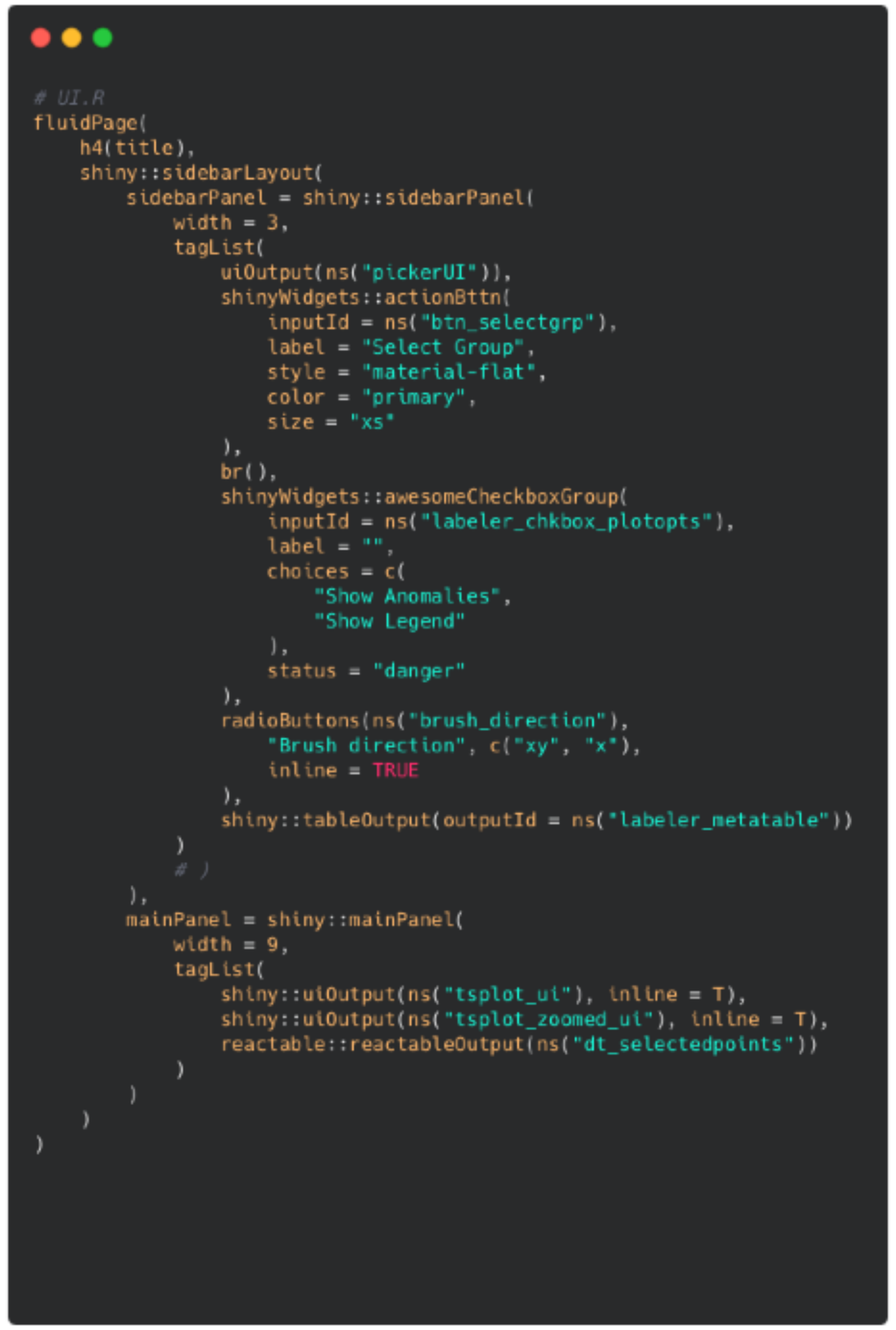
/scalability

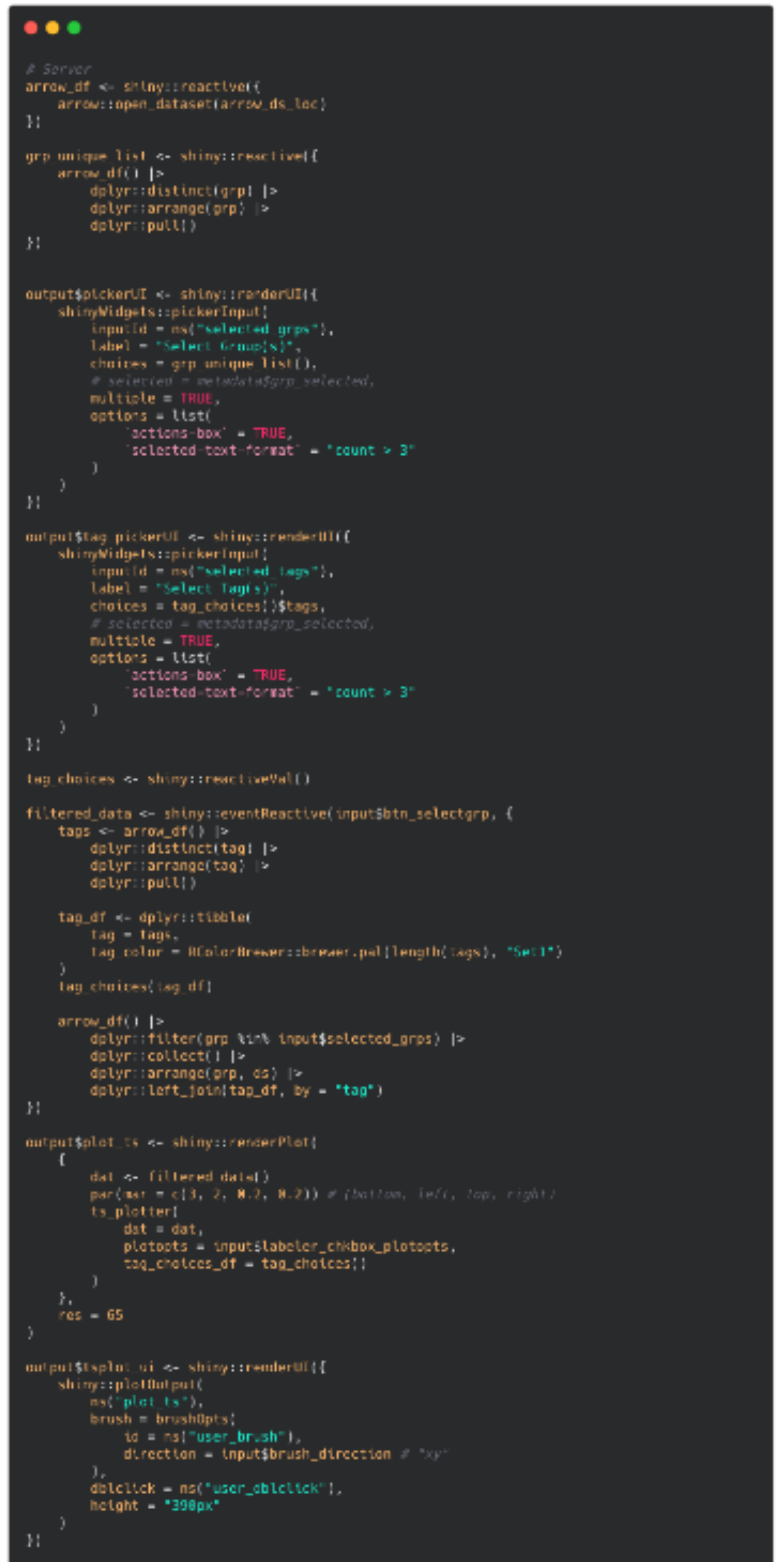
The challenge with simple Shiny

ui.R

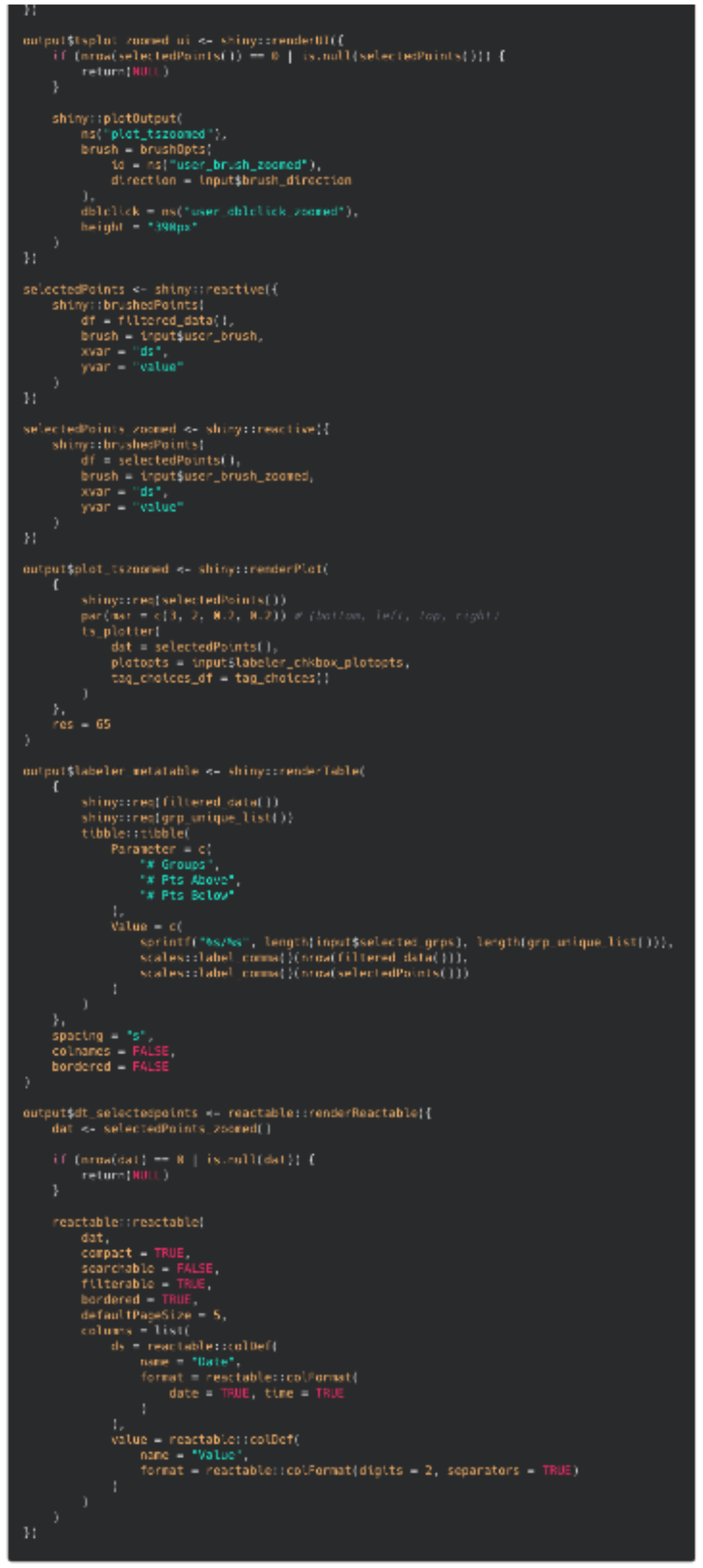


```r
# UI.R
fluidPage(
    h4(title),
    shiny::sidebarLayout(
        sidebarPanel = shiny::sidebarPanel(
            width = 3,
            tagList(
                uiOutput(ns("pickerUI")),
                shinyWidgets::actionBttn(
                    inputId = ns("btn_selectgrp"),
                    label = "Select Group",
                    style = "material-flat",
                    color = "primary",
                    size = "xs"
                ),
                br(),
                shinyWidgets::awesomeCheckboxGroup(
                    inputId = ns("labeler_chkbox_plotopts"),
                    label = "",
                    choices = c(
                        "Show Anomalies",
                        "Show Legend"
                    ),
                    status = "danger"
                ),
                radioButtons(ns("brush_direction"),
                    "Brush direction", c("xy", "x"),
                    inline = TRUE
                ),
                shiny::tableOutput(outputId = ns("labeler_metatable"))
            )
            # )
        ),
        mainPanel = shiny::mainPanel(
            width = 9,
            tagList(
                shiny::uiOutput(ns("tsplot_ui"), inline = T),
                shiny::uiOutput(ns("tsplot_zoomed_ui"), inline = T),
                reactable::reactableOutput(ns("dt_selectedpoints"))
            )
        )
    )
)
```

server.R



```r
# Server
arrow_df <- shiny::reactive({
    arrow::open_dataset(arrow_ds_loc)
})

grp_unique_list <- shiny::reactive({
    arrow_df() |>
        dplyr::distinct(grp) |>
        dplyr::arrange(grp) |>
        dplyr::pull()
})


output$pickerUI <- shiny::renderUI({
    shinyWidgets::pickerInput(
        inputId = ns("selected_grps"),
        label = "Select Group(s)",
        choices = grp_unique_list(),
        # selected = metadata$grp_selected,
        multiple = TRUE,
        options = list(
            `actions-box` = TRUE,
            `selected-text-format` = "count > 3"
        )
    )
})

output$tag_pickerUI <- shiny::renderUI({
    shinyWidgets::pickerInput(
        inputId = ns("selected_tags"),
        label = "Select Tag(s)",
        choices = tag_choices()$tags,
        # selected = metadata$grp_selected,
        multiple = TRUE,
        options = list(
            `actions-box` = TRUE,
            `selected-text-format` = "count > 3"
        )
    )
})

tag_choices <- shiny::reactiveVal()

filtered_data <- shiny::eventReactive(input$btn_selectgrp, {
    tags <- arrow_df() |>
        dplyr::distinct(tag) |>
        dplyr::arrange(tag) |>
        dplyr::pull()

    tag_df <- dplyr::tibble(
        tag = tags,
        tag_color = RColorBrewer::brewer.pal(length(tags), "Set1")
    )
    tag_choices(tag_df)

    arrow_df() |>
        dplyr::filter(grp %in% input$selected_grps) |>
        dplyr::collect() |>
        dplyr::arrange(grp, ds) |>
        dplyr::left_join(tag_df, by = "tag")
})

output$plot_ts <- shiny::renderPlot(
    {
        dat <- filtered_data()
        par(mar = c(3, 2, 0.2, 0.2)) # (bottom, left, top, right)
        ts_plotter(
            dat = dat,
            plotopts = input$labeler_chkbox_plotopts,
            tag_choices_df = tag_choices()
        )
    },
    res = 65
)

output$tsplot_ui <- shiny::renderUI({
    shiny::plotOutput(
        ns("plot_ts"),
        brush = brushOpts(
            id = ns("user_brush"),
            direction = input$brush_direction # "xy"
        ),
        dblclick = ns("user_dblclick"),
        height = "390px"
    )
})
```

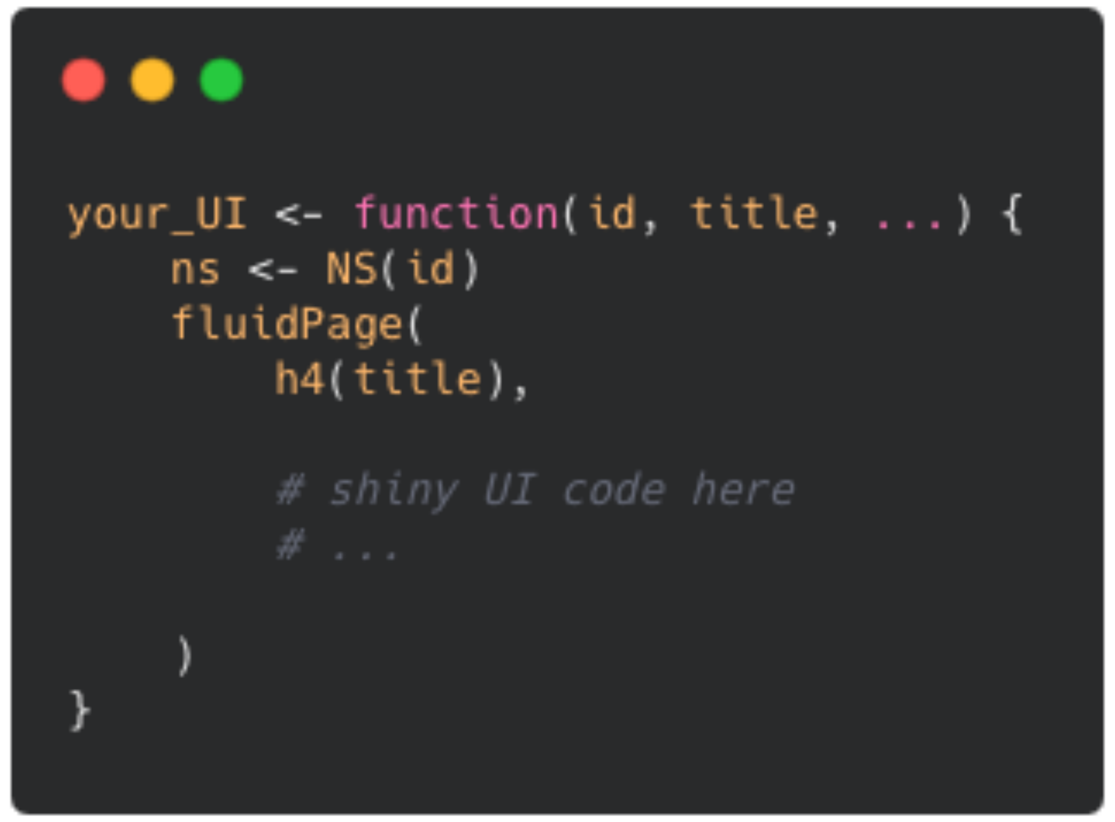


```r
})

output$tsplot_zoomed_ui <- shiny::renderUI({
    if (nrow(selectedPoints()) == 0 | is.null(selectedPoints())) {
        return(NULL)
    }

    shiny::plotOutput(
        ns("plot_tszoomed"),
        brush = brushOpts(
            id = ns("user_brush_zoomed"),
            direction = input$brush_direction
        ),
        dblclick = ns("user_dblclick_zoomed"),
        height = "390px"
    )
})

selectedPoints <- shiny::reactive({
    shiny::brushedPoints(
        df = filtered_data(),
        brush = input$user_brush,
        xvar = "ds",
        yvar = "value"
    )
})

selectedPoints_zoomed <- shiny::reactive({
    shiny::brushedPoints(
        df = selectedPoints(),
        brush = input$user_brush_zoomed,
        xvar = "ds",
        yvar = "value"
    )
})

output$plot_tszoomed <- shiny::renderPlot(
    {
        shiny::req(selectedPoints())
        par(mar = c(3, 2, 0.2, 0.2)) # (bottom, left, top, right)
        ts_plotter(
            dat = selectedPoints(),
            plotopts = input$labeler_chkbox_plotopts,
            tag_choices_df = tag_choices()
        )
    },
    res = 65
)

output$labeler_metatable <- shiny::renderTable(
    {
        shiny::req(filtered_data())
        shiny::req(grp_unique_list())
        tibble::tibble(
            Parameter = c(
                "# Groups",
                "# Pts Above",
                "# Pts Below"
            ),
            Value = c(
                sprintf("%s/%s", length(input$selected_grps), length(grp_unique_list())),
                scales::label_comma()(nrow(filtered_data())),
                scales::label_comma()(nrow(selectedPoints()))
            )
        )
    },
    spacing = "s",
    colnames = FALSE,
    bordered = FALSE
)

output$dt_selectedpoints <- reactable::renderReactable({
    dat <- selectedPoints_zoomed()

    if (nrow(dat) == 0 | is.null(dat)) {
        return(NULL)
    }

    reactable::reactable(
        dat,
        compact = TRUE,
        searchable = FALSE,
        filterable = TRUE,
        bordered = TRUE,
        defaultPageSize = 5,
        columns = list(
            ds = reactable::colDef(
                name = "Date",
                format = reactable::colFormat(
                    date = TRUE, time = TRUE
                )
            ),
            value = reactable::colDef(
                name = "Value",
                format = reactable::colFormat(digits = 2, separators = TRUE)
            )
        )
    )
})
```

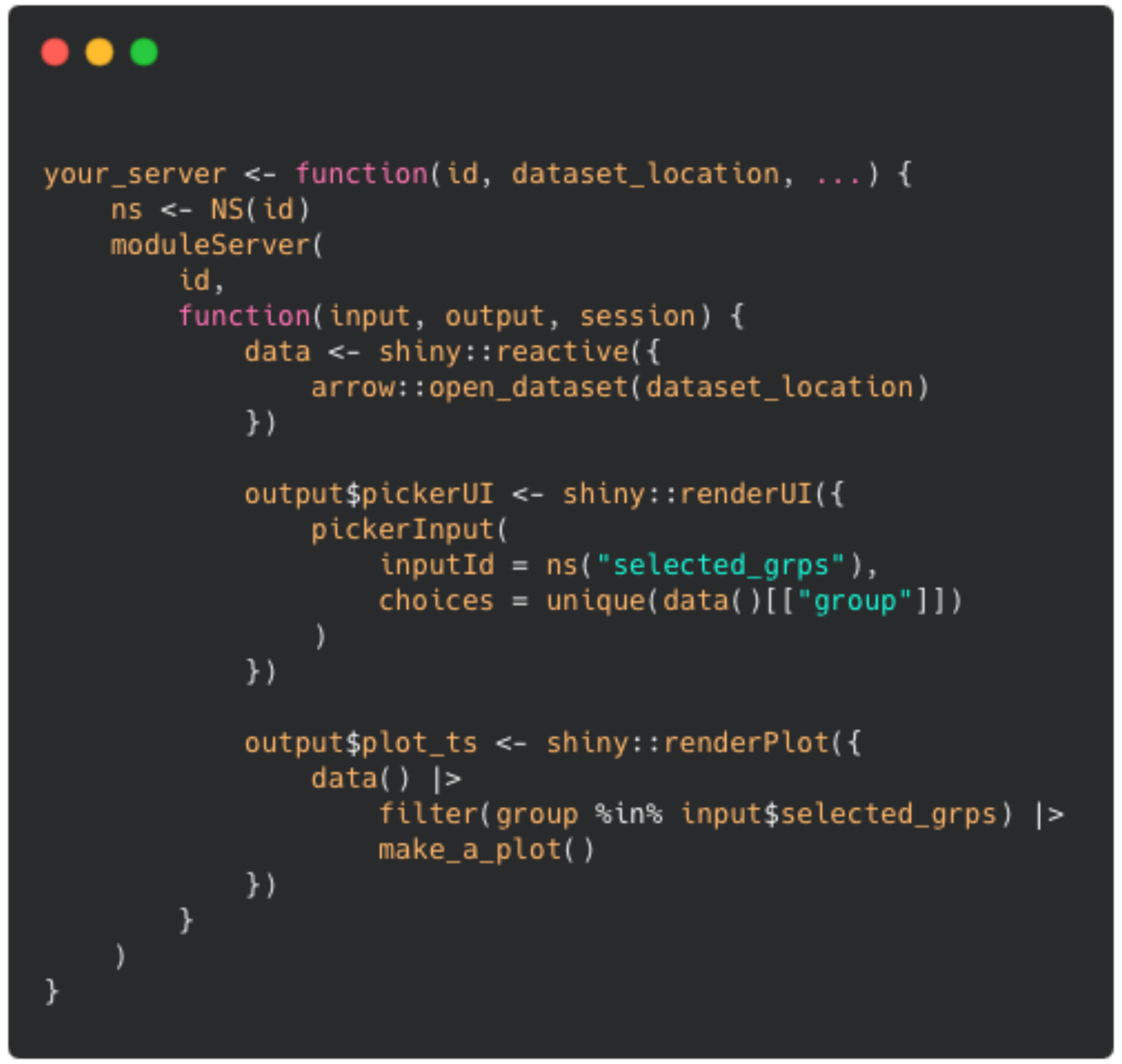
/shiny-modules

The solution to scalability

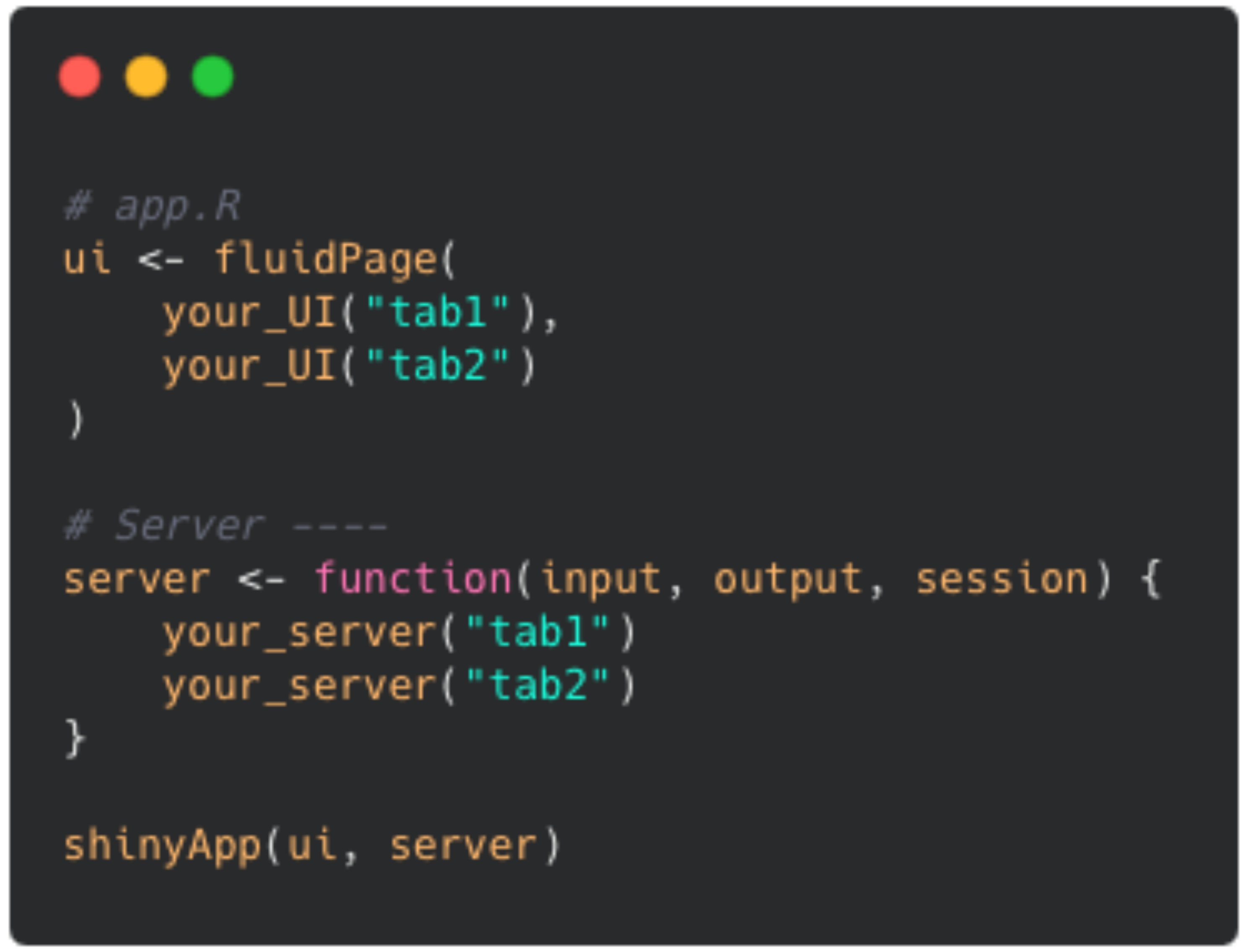
- Modules are *functions* they help you reuse code;
  anything you can do with a function, you can do with a module
- *Namespacing makes it easier*



```r
your_UI <- function(id, title, ...) {
    ns <- NS(id)
    fluidPage(
        h4(title),

        # shiny UI code here
        # ...

    )
}
```



```r
your_server <- function(id, dataset_location, ...) {
    ns <- NS(id)
    moduleServer(
        id,
        function(input, output, session) {
            data <- shiny::reactive({
                arrow::open_dataset(dataset_location)
            })

            output$pickerUI <- shiny::renderUI({
                pickerInput(
                    inputId = ns("selected_grps"),
                    choices = unique(data()[["group"]])
                )
            })

            output$plot_ts <- shiny::renderPlot({
                data() |>
                    filter(group %in% input$selected_grps) |>
                    make_a_plot()
            })
        }
    )
}
```



```r
# app.R
ui <- fluidPage(
    your_UI("tab1"),
    your_UI("tab2")
)

# Server ----
server <- function(input, output, session) {
    your_server("tab1")
    your_server("tab2")
}

shinyApp(ui, server)
```