Meetfy

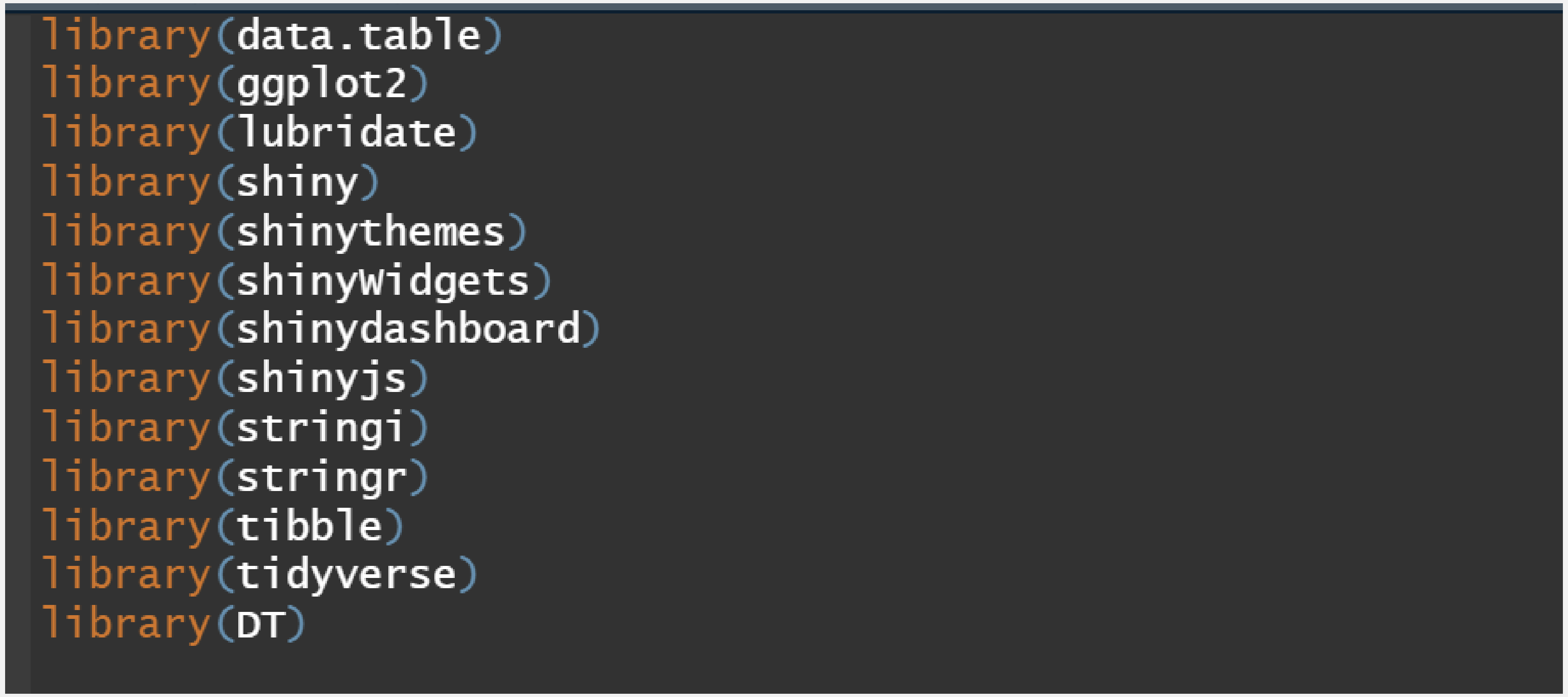
PLAYLISTY

KONCEPCJA

- Pobranie pliku csv przez użytkownika
- Wgranie playlist do naszej aplikacji
- Wybór przez użytkownika parametrów utworów, które mają znaleźć się na ostatecznej playliście (m.in. Data wydania, popularność, długość)
- Analiza utworów i stworzenie statystyk przekazanych playlist
- Pobranie wygenerowanej playlisty z aplikacji

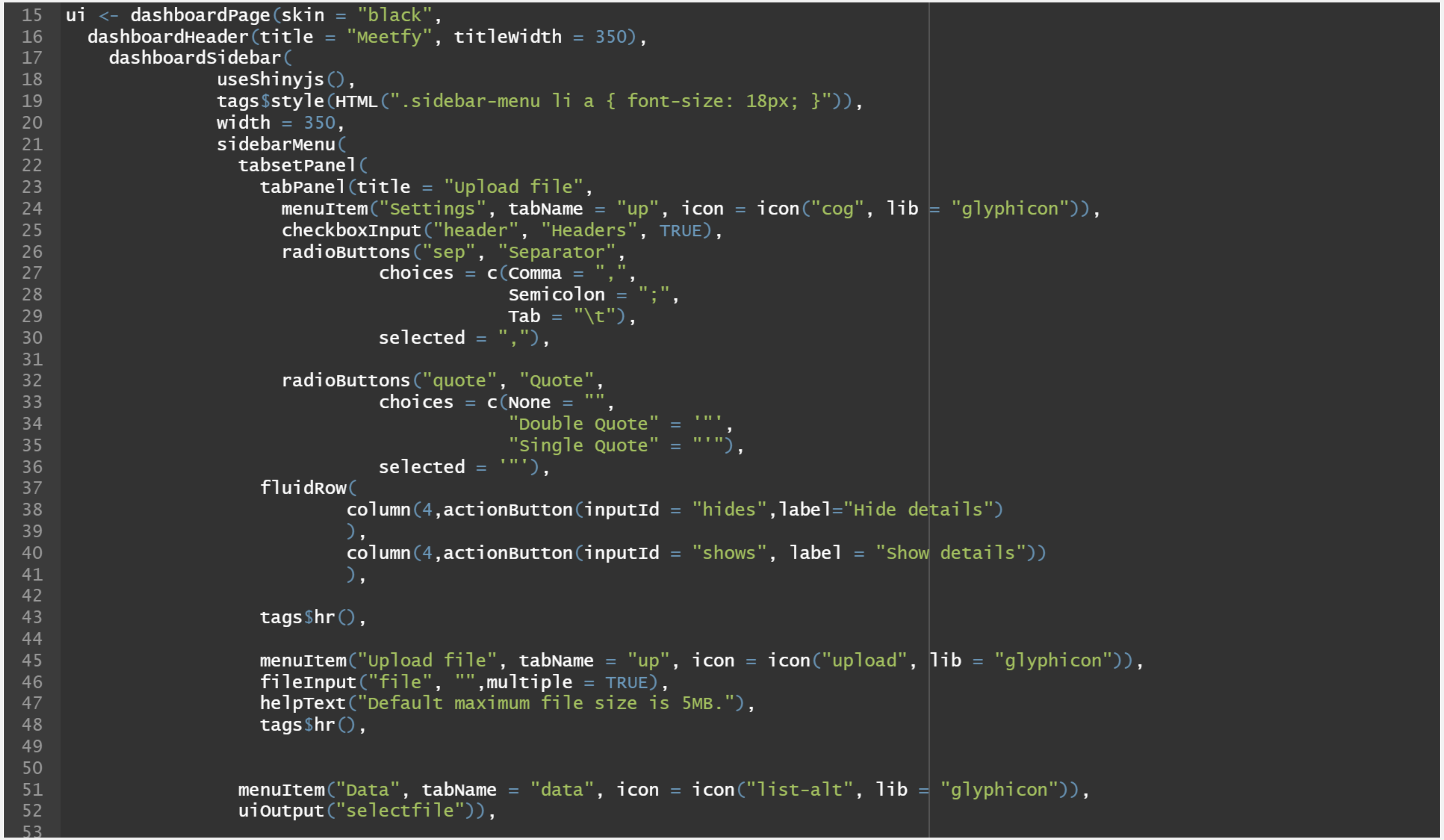
KOD

- Biblioteki



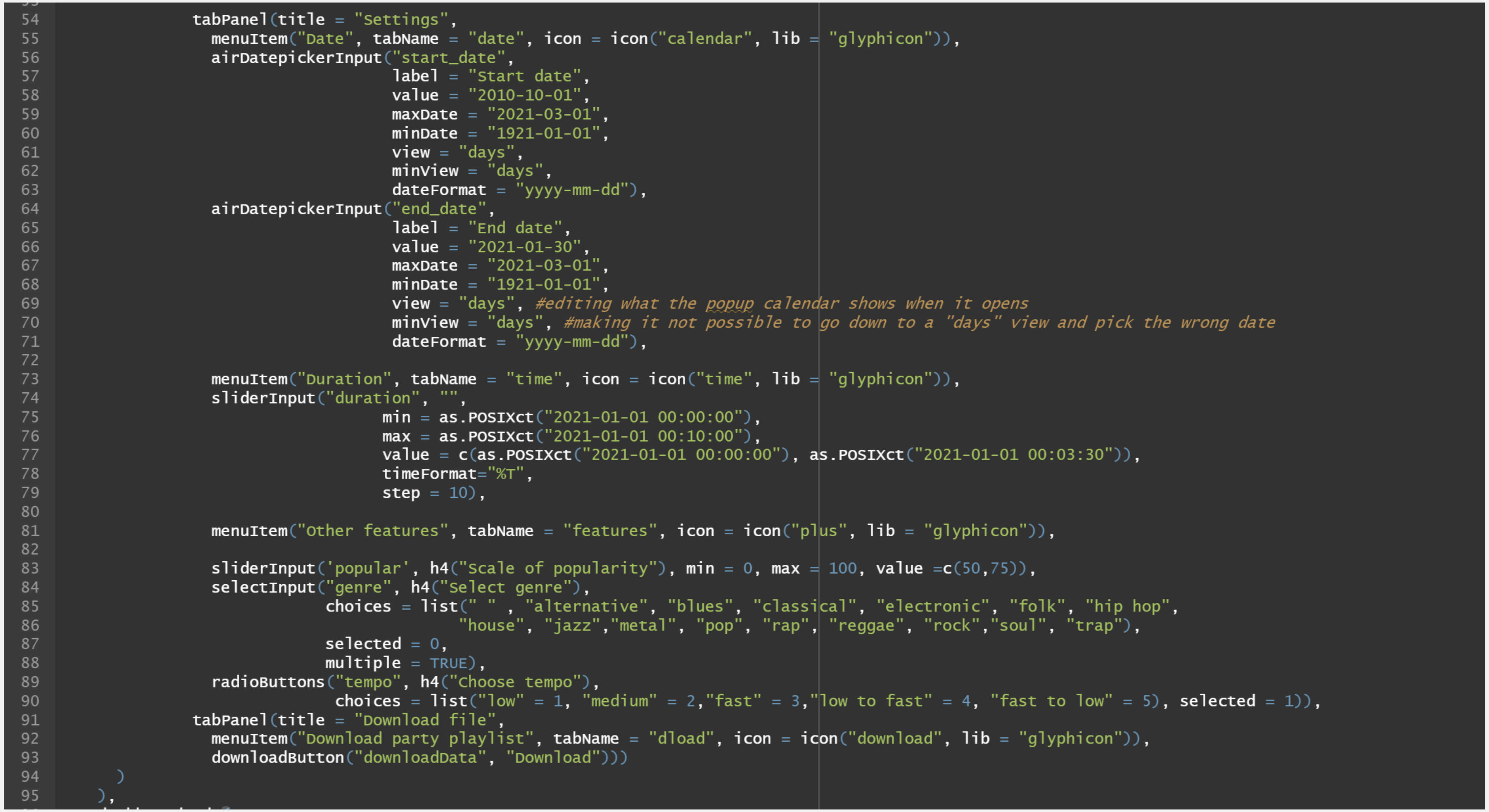
```r
library(data.table)
library(ggplot2)
library(lubridate)
library(shiny)
library(shinythemes)
library(shinyWidgets)
library(shinydashboard)
library(shinyjs)
library(stringi)
library(stringr)
library(tibble)
library(tidyverse)
library(DT)
```

- UI



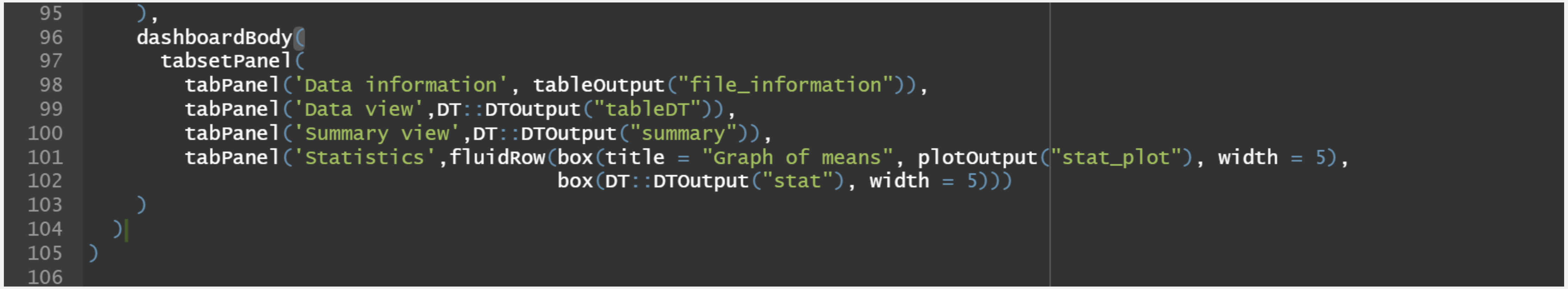
```r
ui <- dashboardPage(skin = "black",
 dashboardHeader(title = "Meetfy", titleWidth = 350),
  dashboardSidebar(
       useShinyjs(),
       tags$style(HTML(".sidebar-menu li a { font-size: 18px; }")),
       width = 350,
       sidebarMenu(
        tabsetPanel(
         tabPanel(title = "Upload file",
          menuItem("Settings", tabName = "up", icon = icon("cog", lib = "glyphicon")),
          checkboxInput("header", "Headers", TRUE),
          radioButtons("sep", "Separator",
              choices = c(Comma = ",",
                    Semicolon = ";",
                    Tab = "\t"),
              selected = ","),

          radioButtons("quote", "Quote",
              choices = c(None = "",
                    "Double Quote" = '"',
                    "Single Quote" = "'"),
              selected = '"'),
         fluidRow(
             column(4,actionButton(inputId = "hides",label="Hide details")
             ),
             column(4,actionButton(inputId = "shows", label = "Show details"))
             ),

         tags$hr(),

         menuItem("Upload file", tabName = "up", icon = icon("upload", lib = "glyphicon")),
         fileInput("file", "",multiple = TRUE),
         helpText("Default maximum file size is 5MB."),
         tags$hr(),


        menuItem("Data", tabName = "data", icon = icon("list-alt", lib = "glyphicon")),
        uiOutput("selectfile")),
```

- UI



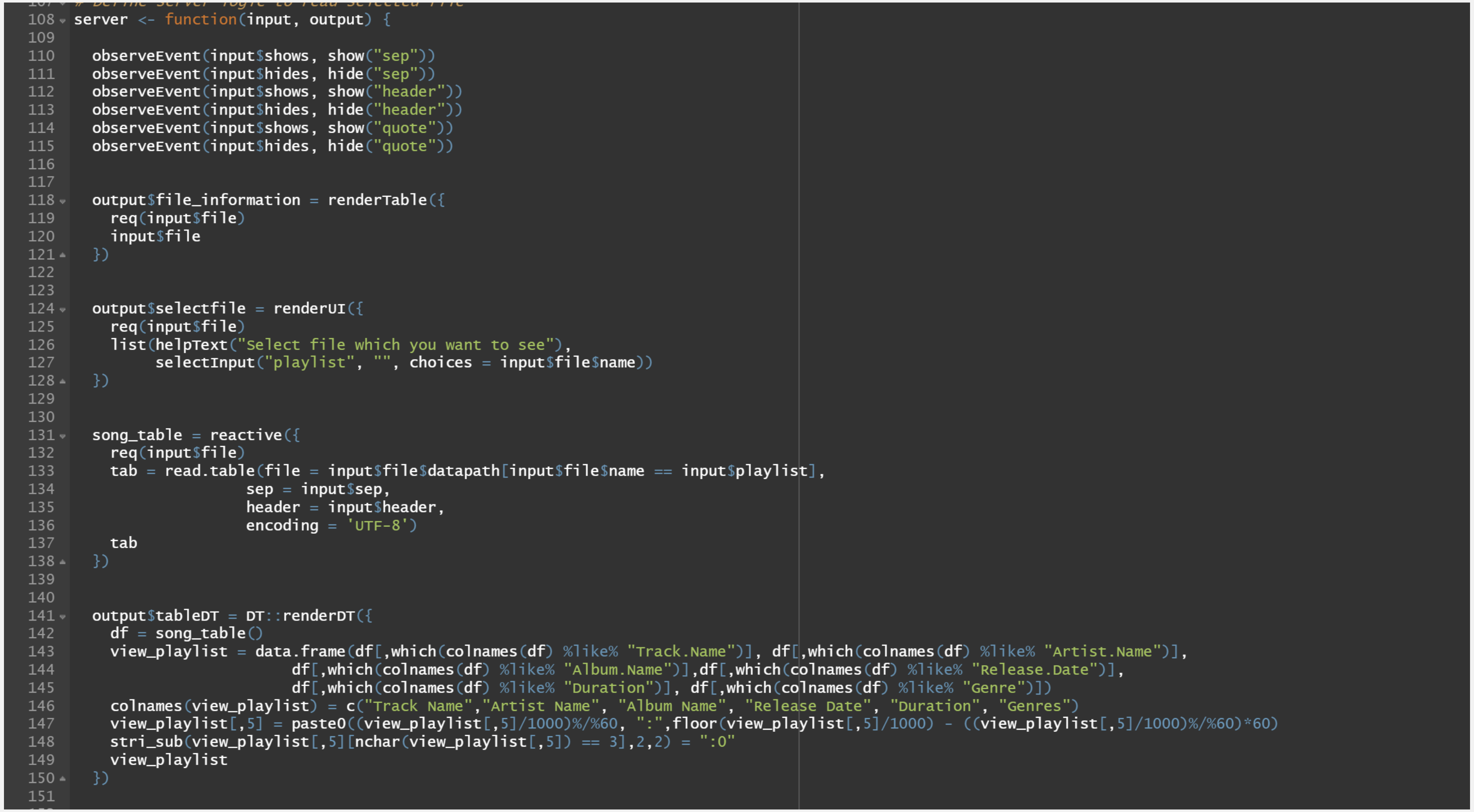
```r
              tabPanel(title = "Settings",
                menuItem("Date", tabName = "date", icon = icon("calendar", lib = "glyphicon")),
                airDatepickerInput("start_date",
                                   label = "Start date",
                                   value = "2010-10-01",
                                   maxDate = "2021-03-01",
                                   minDate = "1921-01-01",
                                   view = "days",
                                   minView = "days",
                                   dateFormat = "yyyy-mm-dd"),
                airDatepickerInput("end_date",
                                   label = "End date",
                                   value = "2021-01-30",
                                   maxDate = "2021-03-01",
                                   minDate = "1921-01-01",
                                   view = "days", #editing what the popup calendar shows when it opens
                                   minView = "days", #making it not possible to go down to a "days" view and pick the wrong date
                                   dateFormat = "yyyy-mm-dd"),

                menuItem("Duration", tabName = "time", icon = icon("time", lib = "glyphicon")),
                sliderInput("duration", "",
                                  min = as.POSIXct("2021-01-01 00:00:00"),
                                  max = as.POSIXct("2021-01-01 00:10:00"),
                                  value = c(as.POSIXct("2021-01-01 00:00:00"), as.POSIXct("2021-01-01 00:03:30")),
                                  timeFormat="%T",
                                  step = 10),

                menuItem("Other features", tabName = "features", icon = icon("plus", lib = "glyphicon")),

                sliderInput('popular', h4("Scale of popularity"), min = 0, max = 100, value =c(50,75)),
                selectInput("genre", h4("Select genre"),
                            choices = list(" " , "alternative", "blues", "classical", "electronic", "folk", "hip hop",
                                          "house", "jazz","metal", "pop", "rap", "reggae", "rock","soul", "trap"),
                            selected = 0,
                            multiple = TRUE),
                radioButtons("tempo", h4("Choose tempo"),
                             choices = list("low" = 1, "medium" = 2,"fast" = 3,"low to fast" = 4, "fast to low" = 5), selected = 1)),
              tabPanel(title = "Download file",
                menuItem("Download party playlist", tabName = "dload", icon = icon("download", lib = "glyphicon")),
                downloadButton("downloadData", "Download")))
      )
    ),
```

- UI



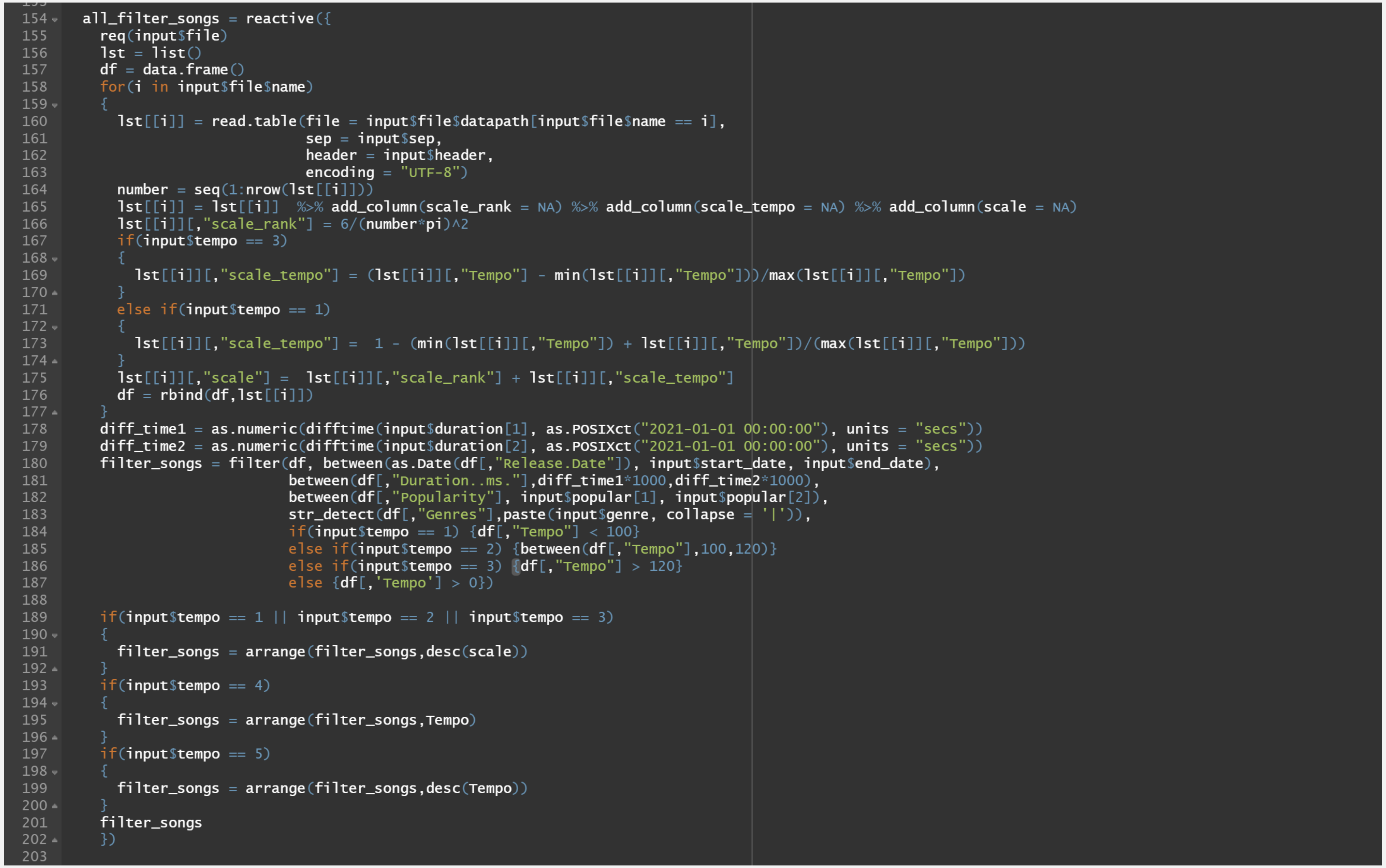
```
    ),
    dashboardBody(
      tabsetPanel(
        tabPanel('Data information', tableOutput("file_information")),
        tabPanel('Data view',DT::DTOutput("tableDT")),
        tabPanel('Summary view',DT::DTOutput("summary")),
        tabPanel('Statistics',fluidRow(box(title = "Graph of means", plotOutput("stat_plot"), width = 5),
                                       box(DT::DTOutput("stat"), width = 5)))
    )
  )
)
```

- Server



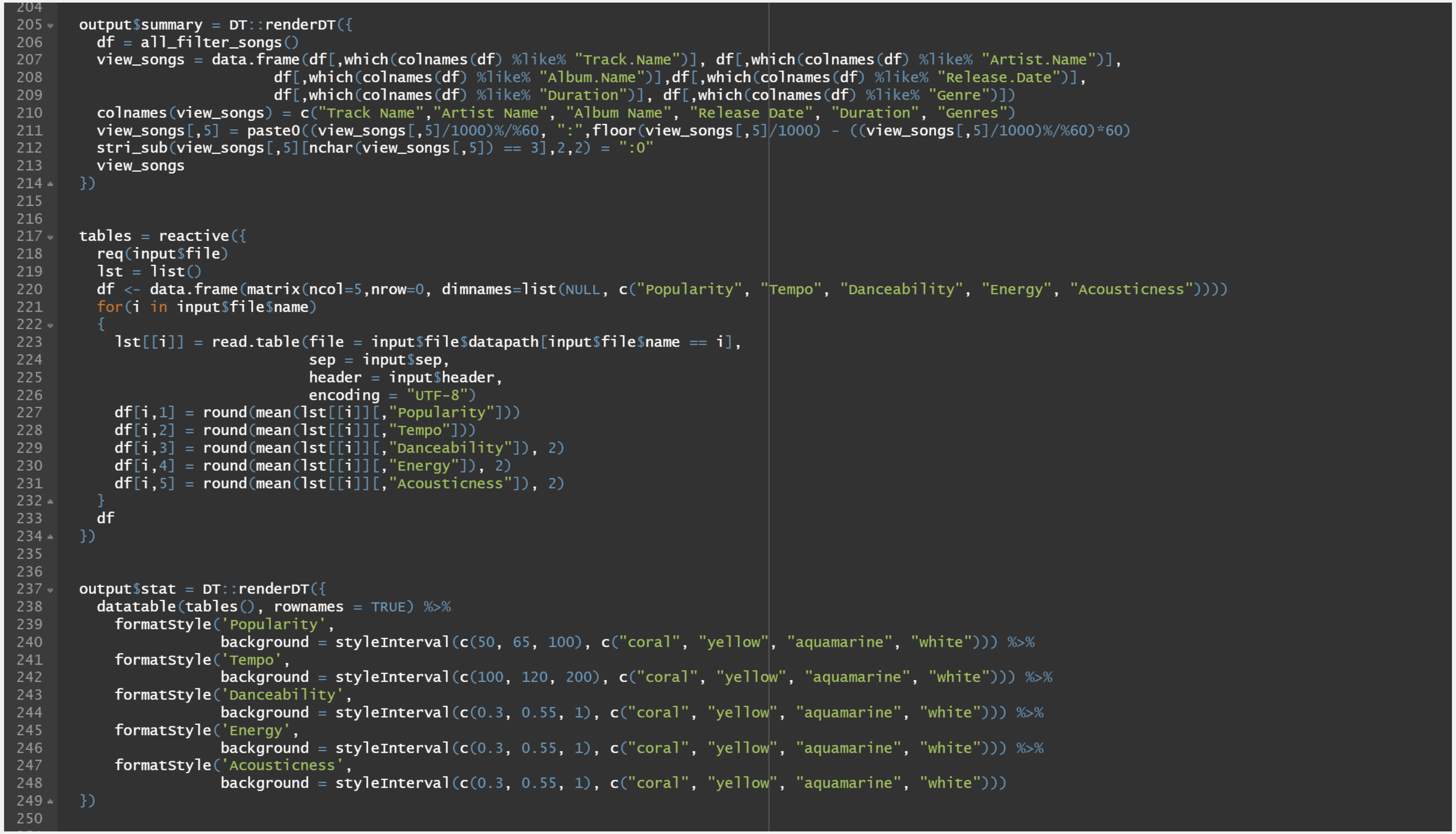
```
# Define server logic to read selected file
server <- function(input, output) {

  observeEvent(input$shows, show("sep"))
  observeEvent(input$hides, hide("sep"))
  observeEvent(input$shows, show("header"))
  observeEvent(input$hides, hide("header"))
  observeEvent(input$shows, show("quote"))
  observeEvent(input$hides, hide("quote"))


  output$file_information = renderTable({
    req(input$file)
    input$file
  })


  output$selectfile = renderUI({
    req(input$file)
    list(helpText("Select file which you want to see"),
         selectInput("playlist", "", choices = input$file$name))
  })


  song_table = reactive({
    req(input$file)
    tab = read.table(file = input$file$datapath[input$file$name == input$playlist],
                     sep = input$sep,
                     header = input$header,
                     encoding = 'UTF-8')
    tab
  })


  output$tableDT = DT::renderDT({
    df = song_table()
    view_playlist = data.frame(df[,which(colnames(df) %like% "Track.Name")], df[,which(colnames(df) %like% "Artist.Name")],
                        df[,which(colnames(df) %like% "Album.Name")],df[,which(colnames(df) %like% "Release.Date")],
                        df[,which(colnames(df) %like% "Duration")], df[,which(colnames(df) %like% "Genre")])
    colnames(view_playlist) = c("Track Name","Artist Name", "Album Name", "Release Date", "Duration", "Genres")
    view_playlist[,5] = paste0((view_playlist[,5]/1000)%/%60, ":",floor(view_playlist[,5]/1000) - ((view_playlist[,5]/1000)%/%60)*60)
    stri_sub(view_playlist[,5][nchar(view_playlist[,5]) == 3],2,2) = ":0"
    view_playlist
  })

```

- Server



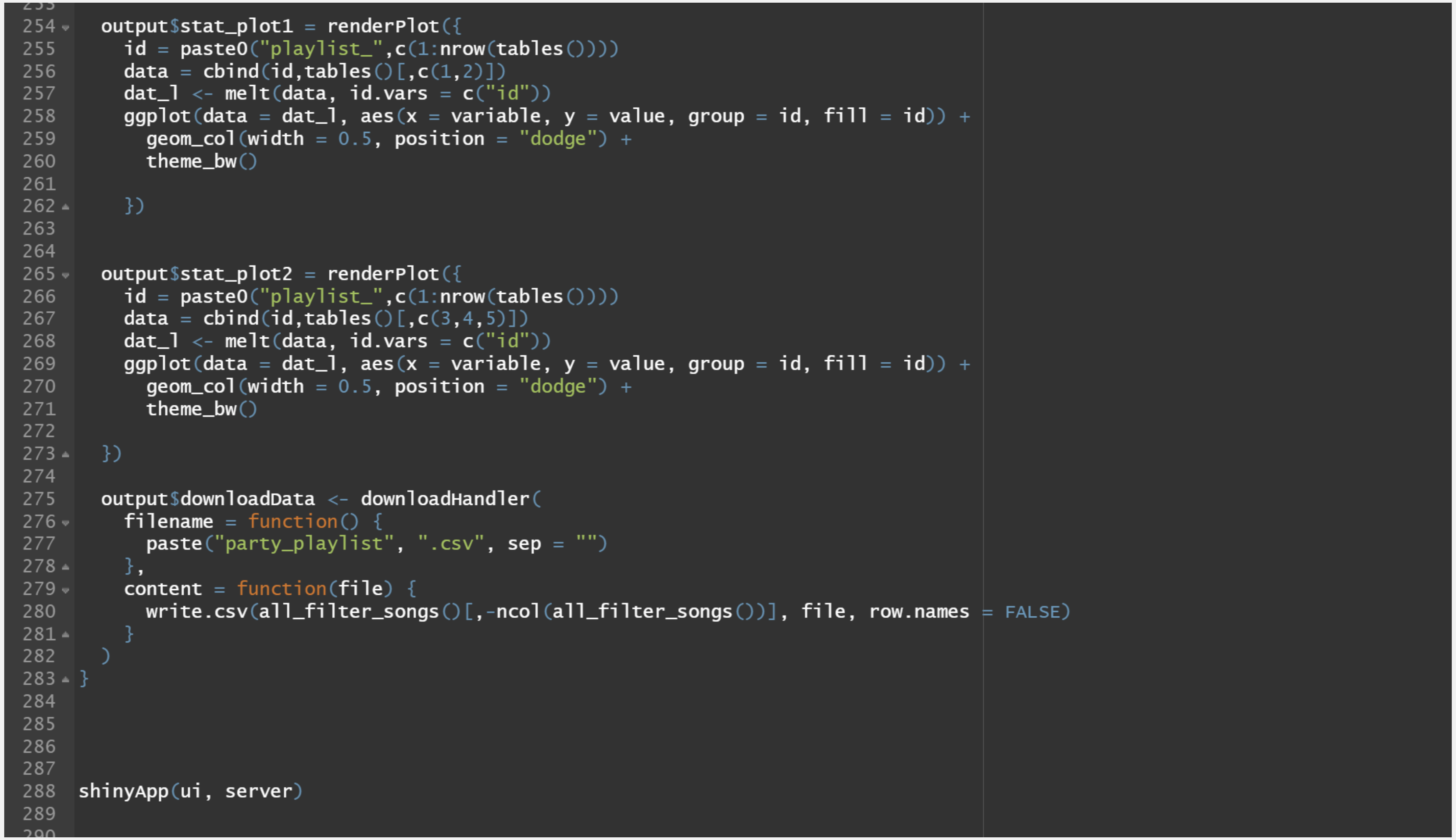
```r
all_filter_songs = reactive({
  req(input$file)
  lst = list()
  df = data.frame()
  for(i in input$file$name)
  {
    lst[[i]] = read.table(file = input$file$datapath[input$file$name == i],
                          sep = input$sep,
                          header = input$header,
                          encoding = "UTF-8")
    number = seq(1:nrow(lst[[i]]))
    lst[[i]] = lst[[i]]  %>% add_column(scale_rank = NA) %>% add_column(scale_tempo = NA) %>% add_column(scale = NA)
    lst[[i]][,"scale_rank"] = 6/(number*pi)^2
    if(input$tempo == 3)
    {
      lst[[i]][,"scale_tempo"] = (lst[[i]][,"Tempo"] - min(lst[[i]][,"Tempo"]))/max(lst[[i]][,"Tempo"])
    }
    else if(input$tempo == 1)
    {
      lst[[i]][,"scale_tempo"] =  1 - (min(lst[[i]][,"Tempo"]) + lst[[i]][,"Tempo"])/(max(lst[[i]][,"Tempo"]))
    }
    lst[[i]][,"scale"] =  lst[[i]][,"scale_rank"] + lst[[i]][,"scale_tempo"]
    df = rbind(df,lst[[i]])
  }
  diff_time1 = as.numeric(difftime(input$duration[1], as.POSIXct("2021-01-01 00:00:00"), units = "secs"))
  diff_time2 = as.numeric(difftime(input$duration[2], as.POSIXct("2021-01-01 00:00:00"), units = "secs"))
  filter_songs = filter(df, between(as.Date(df[,"Release.Date"]), input$start_date, input$end_date),
                        between(df[,"Duration..ms."],diff_time1*1000,diff_time2*1000),
                        between(df[,"Popularity"], input$popular[1], input$popular[2]),
                        str_detect(df[,"Genres"],paste(input$genre, collapse = '|')),
                        if(input$tempo == 1) {df[,"Tempo"] < 100}
                        else if(input$tempo == 2) {between(df[,"Tempo"],100,120)}
                        else if(input$tempo == 3) {df[,"Tempo"] > 120}
                        else {df[,'Tempo'] > 0})

  if(input$tempo == 1 || input$tempo == 2 || input$tempo == 3)
  {
    filter_songs = arrange(filter_songs,desc(scale))
  }
  if(input$tempo == 4)
  {
    filter_songs = arrange(filter_songs,Tempo)
  }
  if(input$tempo == 5)
  {
    filter_songs = arrange(filter_songs,desc(Tempo))
  }
  filter_songs
})
```

- Server



```r
output$summary = DT::renderDT({
  df = all_filter_songs()
  view_songs = data.frame(df[,which(colnames(df) %like% "Track.Name")], df[,which(colnames(df) %like% "Artist.Name")],
                      df[,which(colnames(df) %like% "Album.Name")],df[,which(colnames(df) %like% "Release.Date")],
                      df[,which(colnames(df) %like% "Duration")], df[,which(colnames(df) %like% "Genre")])
  colnames(view_songs) = c("Track Name","Artist Name", "Album Name", "Release Date", "Duration", "Genres")
  view_songs[,5] = paste0((view_songs[,5]/1000)%/%60, ":",floor(view_songs[,5]/1000) - ((view_songs[,5]/1000)%/%60)*60)
  stri_sub(view_songs[,5][nchar(view_songs[,5]) == 3],2,2) = ":0"
  view_songs
})


tables = reactive({
  req(input$file)
  lst = list()
  df <- data.frame(matrix(ncol=5,nrow=0, dimnames=list(NULL, c("Popularity", "Tempo", "Danceability", "Energy", "Acousticness"))))
  for(i in input$file$name)
  {
    lst[[i]] = read.table(file = input$file$datapath[input$file$name == i],
                          sep = input$sep,
                          header = input$header,
                          encoding = "UTF-8")
    df[i,1] = round(mean(lst[[i]][,"Popularity"]))
    df[i,2] = round(mean(lst[[i]][,"Tempo"]))
    df[i,3] = round(mean(lst[[i]][,"Danceability"]), 2)
    df[i,4] = round(mean(lst[[i]][,"Energy"]), 2)
    df[i,5] = round(mean(lst[[i]][,"Acousticness"]), 2)
  }
  df
})


output$stat = DT::renderDT({
  datatable(tables(), rownames = TRUE) %>%
    formatStyle('Popularity',
                background = styleInterval(c(50, 65, 100), c("coral", "yellow", "aquamarine", "white"))) %>%
    formatStyle('Tempo',
                background = styleInterval(c(100, 120, 200), c("coral", "yellow", "aquamarine", "white"))) %>%
    formatStyle('Danceability',
                background = styleInterval(c(0.3, 0.55, 1), c("coral", "yellow", "aquamarine", "white"))) %>%
    formatStyle('Energy',
                background = styleInterval(c(0.3, 0.55, 1), c("coral", "yellow", "aquamarine", "white"))) %>%
    formatStyle('Acousticness',
                background = styleInterval(c(0.3, 0.55, 1), c("coral", "yellow", "aquamarine", "white")))
})
```

- Server



```
  output$stat_plot1 = renderPlot({
    id = paste0("playlist_",c(1:nrow(tables())))
    data = cbind(id,tables()[,c(1,2)])
    dat_l <- melt(data, id.vars = c("id"))
    ggplot(data = dat_l, aes(x = variable, y = value, group = id, fill = id)) +
      geom_col(width = 0.5, position = "dodge") +
      theme_bw()

    })


  output$stat_plot2 = renderPlot({
    id = paste0("playlist_",c(1:nrow(tables())))
    data = cbind(id,tables()[,c(3,4,5)])
    dat_l <- melt(data, id.vars = c("id"))
    ggplot(data = dat_l, aes(x = variable, y = value, group = id, fill = id)) +
      geom_col(width = 0.5, position = "dodge") +
      theme_bw()

  })

  output$downloadData <- downloadHandler(
    filename = function() {
      paste("party_playlist", ".csv", sep = "")
    },
    content = function(file) {
      write.csv(all_filter_songs()[,-ncol(all_filter_songs())], file, row.names = FALSE)
    }
  )
}




shinyApp(ui, server)
```