Beispiel für einen Shiny-Code

Dieses Beispiel erfordert noch Eingabedateien.

|Shiny.R

```
library(shiny)
library(datasets)
library(ggplot2)
require(maptools)

ui <- shinyUI(fluidPage(
  titlePanel("Compartment Model"),
  tabsetPanel(
    tabPanel("Upload Shape File",
             titlePanel("Uploading Map"),
             sidebarLayout(
               sidebarPanel(
                 fileInput(inputId="shpFile", label="Shp",
multiple=TRUE)
                 ),
               mainPanel(
                 plotOutput("map")
               )
             )
        ),

    tabPanel("Upload Rainfall File",
             titlePanel("Uploading Files"),
             sidebarLayout(
               sidebarPanel(
                 fileInput('file1', 'Choose CSV File',
                           accept=c('text/csv',
                                    'text/comma-separated-
values,text/plain',
                                    '.csv')),

                 # added interface for uploading data from
                 # http://shiny.rstudio.com/gallery/file-upload.html
                 tags$br(),
                 checkboxInput('headerR', 'Header', TRUE),
                 radioButtons('sepR', 'Separator',
                              c(Comma=',',
                                Semicolon=';',
                                Tab='\t'),
                              ','),
                 radioButtons('quoteR', 'Quote',
                              c(None='',
                                'Double Quote'='"',
```

```
                                'Single Quote'="'"),
                              '"')

              ),
              mainPanel(
                tableOutput('contentsPrec')
              )
             )
    ),

    tabPanel("Rainfall Plot",
        pageWithSidebar(
            headerPanel('Time Series'),
            sidebarPanel(
                 # "Empty inputs" - they will be updated after the data
is uploaded
                 selectInput('xcol', 'X Variable', "", selected = ""),
                 selectInput('ycol', 'Y Variable', "", selected = "")

            ),
            mainPanel(
                 plotOutput('Plot'),
                 plotOutput('Hist')
            )
        )
    ),

    tabPanel("Climate File",
             titlePanel("Upload Climate Data"),
             sidebarLayout(
               sidebarPanel(
                 fileInput('file2', 'Choose CSV File',
                           accept=c('text/csv',
                                    'text/comma-separated-
values,text/plain',
                                    '.csv')),

                 # added interface for uploading data from
                 # http://shiny.rstudio.com/gallery/file-upload.html
                 tags$br(),
                 checkboxInput('headerC', 'Header', TRUE),
                 radioButtons('sepC', 'Separator',
                              c(Comma=',',
                                Semicolon=';',
                                Tab='\t'),
                              ','),
                 radioButtons('quoteC', 'Quote',
                              c(None='',
                                'Double Quote'='"',
                                'Single Quote'="'"),
                              '"')
```

```
              ),
              mainPanel(
                tableOutput('contentsClim')
              )
            )
    ),

    tabPanel("ETP",
             pageWithSidebar(
               headerPanel('Evaporation'),
               sidebarPanel(
                 # "Empty inputs" - they will be updated after the data
is uploaded
                 # "Empty inputs" - they will be updated after the data
is uploaded
                 selectInput('xcolC', 'X Variable', "", selected = ""),
                 selectInput('ycolC', 'Y Variable', "", selected = ""),
                 sliderInput(inputId = "wc",
                             label = "Wet area %",
                             min = 0.0,
                             max = 5.0,
                             value = 1.5),
                 sliderInput(inputId = "bc",
                             label = "Fetch",
                             min = 0.135,
                             max = 0.25,
                             value = 0.135)
               ),
               mainPanel(
                 plotOutput('PlotC'),
                 plotOutput('HistC')
               )
             )
    ),

    tabPanel("Measured Data",
             titlePanel("Upload Discharge Data"),
             sidebarLayout(
               sidebarPanel(
                 fileInput('file3', 'Choose CSV File',
                           accept=c('text/csv',
                                    'text/comma-separated-
values,text/plain',
                                    '.csv')),

                 tags$br(),
                 checkboxInput('headerM', 'Header', TRUE),
                 radioButtons('sepM', 'Separator',
                              c(Comma=',',
                                Semicolon=';',
```

```
                                Tab='\t'),
                              ','),
                  radioButtons('quoteM', 'Quote',
                              c(None='',
                                'Double Quote'='"',
                                'Single Quote'="'"),
                              '"')

                ),
                mainPanel(
                  tableOutput('contentsMeasured')
                )
              )
    ),

    tabPanel("Runoff model",
          pageWithSidebar(
               headerPanel('Model'),
               sidebarPanel(
                  # "Empty inputs" - they will be updated after the
data is uploaded
                    sliderInput(inputId = "Sia",
                        label = "Initial loss factor",
                        min = 0.05,
                        max = 0.25,
                        value = 0.2),
                    sliderInput(inputId = "Scn",
                        label = "Storage in mm",
                        min = 10,
                        max = 250,
                        value = 50),
                    plotOutput('scsCurveSidebar')
                  ),
               mainPanel(
                 plotOutput('scsCurve')
               )
          )
    ),

    tabPanel("Discharge",
             pageWithSidebar(
               headerPanel('Discharge in qms'),
               sidebarPanel(
                 # "Empty inputs" - they will be updated after the data
is uploaded
                 numericInput("Area", "Basin area in km:",
                              10, min = 1,
                              max = 10000),
                              # verbatimTextOutput("area"),
                 numericInput("Length", "Channel length in km:",
                              10, min = 1,
```

```
                            max = 100),
                            # verbatimTextOutput("length"),
              numericInput("Width", "Channel width in m:",
                            10, min = 1,
                            max = 100),
              sliderInput(inputId = "Slope",
                          label = "Slope in m/m",
                          min = 0.001,
                          max = 0.1,
                          value = 0.01),
              sliderInput(inputId = "Ca",
                          label = "Compartment area sm",
                          value = 1000,
                          min = 0,
                          max = 1000000),
              sliderInput(inputId = "Tl",
                          label = "Transmission loss rate",
                          value = 0.0,
                          min = 0.0,
                          max = 10.0),
            ),
            mainPanel(textOutput("Tcmsi"),
                      textOutput("CIA"),
                      plotOutput('Discharge')
            )
          )
        ),

    tabPanel("Calibration",
             pageWithSidebar(
               headerPanel('Fit model to measured data'),
               sidebarPanel(
                 selectInput('xcolM', 'X Variable', "", selected = ""),
                 selectInput('ycolM', 'Y Variable', "", selected = ""),
                 sliderInput(inputId = "rcp",
                             label = "runoff parameter",
                             min = 0.0,
                             max = 5.0,
                             value = 1.0),
                 sliderInput(inputId = "mtp",
                             label = "threshold parameter",
                             min = 0.0,
                             max = 5.0,
                             value = 1.0)
               ),
               mainPanel(
                 plotOutput('PlotM'),
                 plotOutput('HistM')
               )
             )
          )
```

```
       )
   )
)


server <- shinyServer(function(input, output, session) {
    # added "session" because updateSelectInput requires it

  # rainfall file and processing
  data <- reactive({
    req(input$file1) ## ?req #  require that the input is available

    inFile <- input$file1

    # tested with a following dataset: write.csv(mtcars, "mtcars.csv")
    # and                              write.csv(iris, "iris.csv")
    df <- read.csv(inFile$datapath, header = input$headerR, sep =
input$sepR,
             quote = input$quoteR)


    # Update inputs (you could create an observer with both
updateSel...)
    # You can also constraint your choices. If you wanted select only
numeric
    # variables you could set "choices = sapply(df, is.numeric)"
    # It depends on what do you want to do later on.

    updateSelectInput(session, inputId = 'xcol', label = 'X Variable',
                      choices = "Date", selected = "Date")
    updateSelectInput(session, inputId = 'ycol', label = 'Y Variable',
                      choices = names(df), selected = names(df)[2])

    return(df)
  })

  output$contentsPrec <- renderTable({
      data()
  })

  output$Plot <- renderPlot({
    # plot the data using ggplot
    datumR <- as.Date(data()[, input$xcol],format = "%d/%m/%Y")
    precData <- data()[,input$ycol]
    dx <- data.frame(datumR,precData)
    ggplot(dx, aes(x = datumR, y = precData)) +
      geom_bar(stat="identity") +
      labs(x = "Date",
           y = "Precipitation (mm)",
           title = "Precipitation Data",
           subtitle = "Khan")
```

```
  })

  output$Hist <- renderPlot({
    # histogram
    dy    <- data()[,input$ycol]
    dy[dy < 1] <- NA
    hist(dy, breaks = 25, freq = FALSE, col = 'darkgray', border =
'white')
  })

  # climate file and processing
  dataC <- reactive({
    req(input$file2) ## ?req #  require that the input is available

    inFileC <- input$file2

    # tested with a following dataset: write.csv(mtcars, "mtcars.csv")
    # and                              write.csv(iris, "iris.csv")
    dfC <- read.csv(inFileC$datapath, header = input$headerC, sep =
input$sepC,
                    quote = input$quoteC)

    # Update inputs (you could create an observer with both
updateSel...)
    # You can also constraint your choices. If you wanted select only
numeric
    # variables you could set "choices = sapply(df, is.numeric)"
    # It depends on what do you want to do later on.

    updateSelectInput(session, inputId = 'xcolC', label = 'X Variable',
                      choices = "Date", selected = "Date")
    updateSelectInput(session, inputId = 'ycolC', label = 'Y Variable',
                      choices = names(dfC), selected = names(dfC)[2])

    return(dfC)
  })

  output$contentsClim <- renderTable({
    dataC()
  })

  output$PlotC <- renderPlot({
    # plot the data using ggplot
    datumC <- as.Date(dataC()[, input$xcolC],format = "%d/%m/%Y")
    climData <- dataC()[,input$ycolC]
    dxC <- data.frame(datumC,climData)
    ggplot(dxC, aes(x = datumC, y = climData)) +
      geom_line(size = 1.0) +
      labs(x = "Date",
           y = "Climate variable",
           title = "Climate Data",
```

```
           subtitle = "Khan")
  })

  output$HistC <- renderPlot({
    # histogram
    dyC    <- dataC()[,input$ycolC]
    dyC[dyC < 1] <- NA
    hist(dyC, breaks = 25, freq = FALSE, col = 'darkgray', border =
'white')
  })

  # discharge file and processing
  dataM <- reactive({
    req(input$file3) ## ?req #  require that the input is available

    inFileC <- input$file3

    dfM <- read.csv(inFileC$datapath, header = input$headerM, sep =
input$sepM,
                    quote = input$quoteM)

    # Update inputs (you could create an observer with both
updateSel...)
    # You can also constraint your choices. If you wanted select only
numeric
    # variables you could set "choices = sapply(df, is.numeric)"
    # It depends on what do you want to do later on.

    updateSelectInput(session, inputId = 'xcolM', label = 'X Variable',
                      choices = "Date", selected = "Date")
    updateSelectInput(session, inputId = 'ycolM', label = 'Y Variable',
                      choices = names(dfM), selected = names(dfM)[2])

    return(dfM)
  })

  output$contentsMeasured <- renderTable({
    dataM()
  })

  output$PlotM <- renderPlot({
    # plot the data using ggplot
    datumM <- as.Date(dataM()[, input$xcolM],format = "%d/%m/%Y")
    measuredData <- dataM()[,input$ycolM]
    dxM <- data.frame(datumM,measuredData)
    ggplot(dxM, aes(x = datumM, y = measuredData)) +
      geom_line(size = 1.0) +
      labs(x = "Date",
           y = "Measured discharge in qm",
           title = "Discharge Data",
           subtitle = "Khan")
```

```
  })

  output$HistM <- renderPlot({
    # histogram
    dyM    <- dataM()[,input$ycolM]
    dyM[dyM < 1] <- NA
    hist(dyM, breaks = 25, freq = FALSE, col = 'darkgray', border =
'white')
  })

  # read a shape file +
  uploadShpfile <- reactive({
    if (!is.null(input$shpFile)){
      shpDF <- input$shpFile
      prevWD <- getwd()
      uploadDirectory <- dirname(shpDF$datapath[1])
      setwd(uploadDirectory)
      for (i in 1:nrow(shpDF)){
        file.rename(shpDF$datapath[i], shpDF$name[i])
      }
      shpName <- shpDF$name[grep(x=shpDF$name, pattern="*.shp")]
      shpPath <- paste(uploadDirectory, shpName, sep="/")
      setwd(prevWD)
      shpFile <- readShapePoly(shpPath)
      return(shpFile)
    } else {
      return()
    }
  })

  output$map <- renderPlot({
    if (!is.null(uploadShpfile())){
      plot(uploadShpfile())
    }
  })

  output$area <- renderText({ input$area})
  output$length <- renderText({ input$length })

  # render SCS plot
  output$scsCurve <- renderPlot({
    # SCS
    lambda <- input$Sia
    scn  <- input$Scn
    sia <- lambda * scn

    # Rainfall to Runoff
    prDatum <- as.Date(data()[, input$xcol],format = "%d/%m/%Y")
    prSeries <- data()[,input$ycol]
    pt <- data.frame(prDatum,prSeries)
```

```
  qsSeries <- prSeries*0

  ip <- length(qsSeries)

  i=1
  while (i<=ip) {
    if (prSeries[i]<=sia) {
      qsSeries[i] <- 0
    } else {
      qsSeries[i] <- (prSeries[i]-sia)^2/(prSeries[i]-sia+scn)
    }
    i = i+1
  }

  qt <- data.frame(prDatum,qsSeries)

  ggplot(qt, aes(x = prDatum, y = qsSeries)) +
      geom_bar(stat="identity") +
      labs(x = "Date",
           y = "Runoff (mm)",
           title = "Runoff Series in mm",
           subtitle = "Khan")

})

# render SCS plot
output$scsCurveSidebar <- renderPlot({
  # SCS
  lambda <- input$Sia
  scn  <- input$Scn
  sia <- lambda * scn
  prs <- seq(0,100,5)
  qss <- prs*0
  ln <- length(prs)

  i=1
  while (i<=ln) {
    if (prs[i]<=sia) {
      qss[i] <- 0
    } else {
      qss[i] <- (prs[i]-sia)^2/(prs[i]-sia+scn)
    }
    i = i+1
  }
  xy <- data.frame(prs,qss)
  ggplot(data=xy, aes(x=prs, y=qss, group=1)) +
    geom_line(linetype = "dashed",color = "red") +
    geom_point() +
    labs(x = "Precipitation per square m in mm",
         y = "Surface runoff per area produced in mm",
         title = "Runoff-Precipitation Response",
```

```
          subtitle = "SCS Model")
  })

  # Calculate Discharge
  output$Discharge <- renderPlot({

    lambda <- input$Sia
    scn    <- input$Scn
    sia    <- lambda * scn

    # CIA
    barea    <- input$Area
    blength  <- input$Length
    bwidth   <- input$Width
    bslope   <- input$Slope
    casm     <- input$Ca
    tlr      <- input$Tl

    # Rainfall to Runoff
    dDatum <- as.Date(data()[, input$xcol],format = "%d/%m/%Y")
    pSeries <- data()[,input$ycol]
    dt <- data.frame(dDatum,pSeries)

    dSeries <- pSeries*0
    dHeight <- pSeries*0
    dLosses <- pSeries*0

    id <- length(dSeries)

    tcmsi    <- 0.0195*(blength*1000)^0.77*bslope^-0.385 # time in
minutes
    tdurs    <- tcmsi*60*2.67 # event duration in hours
    # transmission losses in channel
    trlos    <- blength*1000*bwidth*tlr/1000*tdurs/3600*1/100

    i=1
    while (i<=id) {
      if (pSeries[i]<=sia) {
        dSeries[i] <- 0
      } else {
        # Runoff calculation in dSeries: From mm in pSeries to cubic
meters per second
        # p (mm/sm*d) -> r (mm/sm*d) -> q (cubic m/s): p x area*1E+6 *
1/1000 * 1/86400
        conversion <- barea*1E+6 * 1/1000 * 1/86400
        dSeries[i] <- ((pSeries[i]-sia)^2/(pSeries[i]-
sia+scn))*conversion

        # convolution and routing

        # Previous calculation of losses:
```

```
        # units x Area x infiltration per hour x Duration in hours
        if(dSeries[i] >= trlos){
          dSeries[i]  <- dSeries[i]-trlos
          dLosses[i]  <- trlos
        } else {
          dLosses[i]  <- dSeries[i]
          dSeries[i]  <- 0
        }
      }
      dHeight[i] <- dSeries[i]/bwidth
      i = i+1
    }

    dt <- data.frame(dDatum,dSeries,dLosses,dHeight)
    ggplot(data=dt, aes(x=dDatum, y=dSeries)) +
      geom_line(linetype = "dashed",color = "blue") +
      labs(x = "Date",
           y = "Discharge in qm/s",
           title = "Discharge",
           subtitle = "Model")
  })

  # Calculate Concentration time
  output$Tcmsi <- renderText({
    # CIA
    barea    <- input$Area
    blength  <- input$Length
    bwidth  <- input$Width
    bslope   <- input$Slope
    mn       <- input$Mn

    # concentration time in min/hours
    tcmsi    <- 0.0195*(blength*1000)^0.77*bslope^-0.385 # time in
minutes
    tcmsi    <- format(tcmsi, digits = 2)

    paste("The concentration time of floods is", tcmsi, "minutes.")

  })

  # Calculate Peak Discharge
  output$CIA <- renderText({
    # CIA
    barea    <- input$Area
    blength  <- input$Length
    bwidth  <- input$Width
    bslope   <- input$Slope
    mn       <- input$Mn

    # concentration time in min/hours
    CIA    <- 1/3.6*0.02*10*barea
```

```
    CIA    <- format(CIA, digits = 3)

    paste("A reference event of 10 mm/hour produces a peak discharge of
", CIA, "qm/s.")

  })

})

shinyApp(ui, server)
```