viewTRiAL Package - A technical guide

Suvitha Subramaniam & Katharina Klatte

2020-09-28

Contents

1 About viewTRiAL

The viewTRiAL package is a R Shiny (package {shiny} version 1.5.0) web application prototype designed to build and deploy Centralized Monitoring R shiny web applications in clinical studies within the Department of Clinical Research at University of Basel.

2 Install viewTRiAL

2.1 Installing from CTU's repository

```
setwd("~repos/proj/piCentralizedMonitoring/R")
system("R CMD INSTALL viewTRiAL")
```

2.2 Installing from GitHub

```
library(devtools)
devtools::install_github("suvi-subra/viewTRiAL")
```

2.3 Running viewTRiAL

```
library(viewTRiAL)
run_app()
```

3 Let's get started!

The prototype has been developed as Shiny modules to address namespacing problems in Shiny User interface (UI) and server logic and to facilitate working through huge amount of code. In order to develop a production-grade Shiny App, the framework from the {golem} package version 0.2.1 was used. The structure of a golem framework is the same as that of a typical R package except that it allows for simplified development and deployment of packaged R Shiny applications.

3.1 Load, prepare and save your data (data-raw/)

For maximum efficiency in running your app, each time you launch it, you do not want it to be loaded more than once. Since we are packaging our R Shiny app, we can take advantage of storing data as internal data, making it unavailable to users. A script that loads, prepares and saves the data as internal data is stored under the data-raw/ folder. This script does not run automatically when the app is launched. Therefore, a line to run the script can be incorporated in the app_server.R file.

We use 4 dummy datasets in this prototype app. The structure of each dataset is displayed.

3.1.1 Baseline data

This is baseline data containing one row for each patient. The pat_id, centre, withdrawal status, randomization date and informed consent dates are included.

```
str(dummy_baseline)
#> 'data.frame':    100 obs. of  5 variables:
#>  $ pat_id           : int  100 51 13 40 90 50 31 79 7 61 ...
#>  $ centre           : Factor w/ 5 levels "A","B","C","D",..: 4 3 5 3 2 5 2 2 3 1 ...
#>  $ withdrawn        : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
#>  $ date_random.date : Date, format: "2019-08-15" "2019-08-23" "2020-04-28" "2020-06-04" ...
#>  $ date_consent.date: Date, format: "2019-08-15" "2019-08-23" "2020-04-28" "2020-06-04" ...
```

3.2 Track your development (dev/)

The dev/ folder is used as a notebook to track steps of the development process. There are 3 files:

3.2.1 01_start.R

This file should be filled at the start. It contains functions to fill the DESCRIPTION file, set options and list dependencies.

3.2.2 02_dev.R

This file should be filled to set the structure of core functionalities such as creating modules and other R functions. One is able to create a new module (eg.mod_overview.R) by calling golem::add_module(name = “overview”) and a R file (eg.filename.R) by calling usethis::use_r(“filename”).

3.2.3 03_deploy.R

Finally, this file contains functions that aid in deploying the app via Rstudio or Docker. This file contains functions to create either an app.R file or Docker file to deploy the app depending on the portal you choose. This is a necessary step as the R Shiny web application is packaged.

3.3 Start developing (R/)

Core functions of the app are stored under the R/ folder.

3.3.1 app_ui.R

This function displays the input from its counterpart app_server(). Below is an example of how a menu item in the sidebar of a dashboard is created and its corresponding module called to display the content on the dashboard body.

```
app_ui <- function(request) {

  fluidPage(
    dashboardPage(skin = "purple",
```

```
      dashboardHeader(title = "Study name", titleWidth = 300),
      dashboardSidebar(
        sidebarMenu(
          ## 1. Create a new menu item here. List tabName in get_modules.R.
          menuItem("Overview", tabName = mod$overview, icon = icon("chart-pie"))
        )
      ),
      dashboardBody(
        tabItems(
          ## 2. Call the module corresponding to the menu item here
          mod_overview_ui(id = mod$overview, label = mod$overview)
        )
      )

    )
  )
}
```

There are 4 items to adapt in this file:

1. dashboardHeader(title): Change the study name
2. dateRangeInput(inputId): Replace the name of the date filter depending on the date used to filter the data. (eg. Enrolment date). This filter may be commented if not needed.
3. dateRangeInput(label): Replace the date ranges for filtering.
4. selectInput(choices, selected): Replace the vector of centers and default center to be selected. This filter may be commented if not needed.

```
app_ui <- function(request) {

    ## Get module label names
    mod <- get_modules()

    ## Leave this function for adding external resources
    golem_add_external_resources()
    ## List the first level UI elements here
    fluidPage(
      dashboardPage(skin = "purple",
      ## Header
      ## TODO (1): Change study name
      dashboardHeader(title = "Study name", titleWidth = 300),
      ## Sidebar
      dashboardSidebar(width = 300,
        sidebarMenu(
            ## TAB 1: Overview
            menuItem("Overview", tabName = mod$overview, icon = icon("chart-pie")),
            ## TAB 2: Performance measures
            menuItem("Performance measures", icon = icon("chart-pie"),
              ## TAB 2.1: Recruitment and retention
              menuSubItem("Recruitment and Retention", tabName = mod$recruitment),
              ## TAB 2.2: Informed consent and eligibility
              menuSubItem("Informed consent and Eligibility", tabName = mod$consent),
              ## TAB 2.3: Data quality
              menuSubItem("Data Quality", tabName = mod$quality)),
            ## TAB 3: Study management
```

```
            menuItem("Study Management", icon = icon("chart-pie"),
              ## TAB 3.1: Visits
              menuSubItem("Visits", tabName = mod$visits),
              ## TAB 3.2: Biosampling
              menuSubItem("Biosampling/Imaging data", tabName = mod$lab),
              ## TAB 3.3: Safety
              menuSubItem("Safety management", tabName = mod$safety),
              ## TAB 3.4: Staff management
              menuSubItem("Staff management", tabName = mod$staff)),
            ## FILTER: Date range
            ## TODO (2): Replace 2nd argument "Randomization date:"
            dateRangeInput(inputId =  "period", label = "Randomization date:",
              ## TODO (3): Replace date ranges
              start = as.POSIXct("2019-06-01"),
              end   = as.POSIXct(today())),
            ## FILTER: Center
            ## TODO (4): Replace 3rd argument "choices" with a vector of centers
            selectInput("center", "Center",
                        choices = c("All", sort(levels(dummy_baseline$centre))),
                        selected = "All"),
              width = "350")),
        ## Body
        dashboardBody(
          tabItems(
            ## TAB 1: Overview
            mod_overview_ui(mod$overview, label = mod$overview),
            ## TAB 2.1: Recruitment and retention
            mod_recruit_ui(mod$recruitment, label = mod$recruitment),
            ## TAB 2.2: Informed consent and eligibility
            mod_consent_ui(mod$consent, label = mod$consent)
                  )
        ) ## dashboardBody
      ) ## dashboardPage
    ) ## fluidPage

}
```

3.3.2 app_server.R

This file contains the server logic. In this file, reactive dataframes are created and a callModule() corresponding to the module UI created in app_ui.R is called. Starting in Shiny 1.5.0, it is recommended to use moduleServer() instead of callModule() as it can be tested with testServer(). However, in our prototype since we have additional parameters to pass to each module function, we will stick to using callModule() as moduleServer() will not be able to handle that.

There are 4 items to adapt in this file:

1. The input dataframes in both filter functions and callModule(data1)
2. The variable names of centre and filter date

```
#' The application server-side
#'
#' @param input,output,session Internal parameters for {shiny}.
#'     DO NOT REMOVE.
```

```r
#' @import shiny
#' @noRd
app_server <- function(input, output, session) {

  ## Get module label names
  mod <- get_modules()

  ## Create reactive dataframes
  bl_period <- reactive(

    if(input$center != "All"){
      ## TODO (1): Replace input dataframe
      ## TODO (2): Change variable names "centre" and "date.random.date"
      filter(df = dummy_baseline,
             centre == input$center &
               date_random.date >= input$period[1] &
               date_random.date <= input$period[2])

    } else{
      ## TODO (3): Replace input dataframe
      ## TODO (4): Change variable names "centre" and "date.random.date"
      filter(df = dummy_baseline,
             date_random.date >= input$period[1] &
               date_random.date <= input$period[2])
    })

  # List the first level callModules here
  ## Overview tab
  ## TODO (5): Replace input dataframe in argument data1
  callModule(mod_overview_server, mod$overview, data1 = bl_period)
  callModule(mod_recruit_server, mod$recruitment, data1 = bl_period)
  callModule(mod_consent_server, mod$consent, data1 = bl_period)

}
```

3.3.3 app_config.R

This file contains a get_golem_config() that retrieves app config. No changes to make here.

3.3.4 get_modules.R

This file contains a list of module label names. No changes to make here unless you add a new module.

3.3.5 mod_overview.R

This file contains the ui and server functions for the "Overview" tab. The "Overview" tab contains the following:

- value boxes displaying the number of patients, randomized, currently active and withdrawn
- line plot displaying recruitment over time by centers

Here there are no changes required unless one wants to add value boxes for additional figures or plots.

```r
#' overview UI Function
#'
#' @description A shiny Module.
#'
#' @param id,input,output,session Internal parameters for {shiny}.
#'
#' @noRd
#'
#' @importFrom shiny NS tagList
mod_overview_ui <- function(id, label){
  ns <- NS(id)

    tabItem(tabName = label,
            fluidRow(
              ## No.of participants randomized
              valueBoxOutput(ns("randomized"), width = 12)
            ),

            fluidRow(
              valueBoxOutput(ns("active"), width = 6),
              valueBoxOutput(ns("withdrew"), width = 6)
            ),

            fluidRow(
              ## Plot displaying no.of participants randomized across acute centers
              box(
                width = 6,
                height = "600",
                title = "Recruitment by centers",
                status = "primary",
                plotlyOutput(ns('recruitplot'), height = "500"),
                solidHeader = TRUE,
                collapsible = FALSE),

            )
    )

}

#' overview Server Function
#'
#' @noRd
mod_overview_server <- function(input, output, session, baseline.data){
  ns <- session$ns

  output$randomized <- renderValueBox({
    valueBox(value = nrow(baseline.data()), subtitle = "Randomized", color = "green")
  })

  output$withdrew <- renderValueBox({

    ## Fill in the value of withdrawn
    no <- baseline.data() %>% filter(withdrawn == TRUE) %>% nrow()
```

```
    valueBox(value = no, subtitle = "Withdrawn", color = "red")
  })

  output$active <- renderValueBox({
    no <- baseline.data() %>% filter(withdrawn == FALSE) %>% nrow()
    valueBox(value = no, subtitle = "Currently active", color = "blue")
  })

  output$recruitplot <- renderPlotly({

    ## Enrolment plot data
    plot_recruitment(baseline.data(), "date_random.date", "centre")


  })
}
```

3.3.6 mod_recruit.R

3.3.7 mod_consent.R

3.3.8 plot_recruitment.R

3.3.9 run_app.R