Introduction

VisualSHIELD is a shiny app module. A shiny app module is a self-contained UI with it's own logic that can be easily integrated in any other custom shiny app.

The VisualSHIELD module allows to seamlessly analyze multiple remote datasets in parallel hosted on Opal and optionally provides also a facility to load dbNP data into Opal. The analysis is performed through the privacy-aware DataSHIELD analysis package, and allows to easily perform:

- histograms
- contour plots
- heatmaps
- linear models (lm)
- generalized linear models (glm)

VisualSHIELD module is exposed through the VisualSHIELDUI and the VisualSHIELDServer functions.

Embed the UI module in a custom shiny app

Embedding the UI in your custom shiny app is as simple as using VisualSHIELDUI in your page as any other shiny object. Below a simple example of a minimal custom shiny UI that embeds VisualSHIELDUI.

```r
library(shiny)

library(opalr)
#> Loading required package: httr
library(DSI)
#> Loading required package: progress
#> Loading required package: R6
library(dsBaseClient)
library(VisualSHIELD)

shinyUI(
  fluidPage(
    fluidRow(column(10, uiOutput("server")),
             column(2, actionButton("load", "Update"), tags$style(type='text/css',
                        "#load { vertical-align: middle; margin-top: 25px;}"))),
    fluidRow(VisualSHIELDUI("VisualSHIELD", h3("Demo VisualSHIELD app")))
    )
)
```

you can choose whatever id, and title you like, beware that the choosen ID should be the same as the one passed to the server module.

Embed the Server module in a custom shiny app

Each shiny module also has a server counterpart for the UI. The VisualSHIELDServer communicates with the parent custom app through the servers parameter, this means that we expect it to be a reactive block, returning a list (more on this below) or NULL.

Here is an example on how to embed it in your custom shinyServer function

```
shinyServer(function(input, output, session) {
  # login information, list of servers and user name
  login <- reactive({
    if( is.null(input$load) || !input$load )
      return(NULL)

    isolate(
      list(username="tomasoni", email="tomasoni@cosbi.eu",
                 servers=list(
                   # server 1
                   list(
                           opal_server = list(id = "1",
                                              name = "DEMO",
                                              url = input$custom_server,
                                              username = "administrator",
                                              password = "password",
                                              certificate = NULL,
                                              private_key = NULL),
                           # dbNP server whose studies will be migrated
                           # to the opal server defined above
                           dashin_server = NULL
                   )
                   #, ... server n
                 )
               )
      )
    })

  VisualSHIELDServer("VisualSHIELD", servers=login)

  output$server <- renderUI({
    textInput("custom_server",
              label="Server to connect to:",
              value="",
              placeholder = "https://opal-demo.obiba.org")
  })
})
```

As you can notice, `login` is a reactive block that gets updated every time the `Update` button is clicked. This update in turns triggers the update of VisualSHIELDServer `Dataservers` and the module will re-connect to the updated list of servers. This demonstrates how you can reactively update VisualSHIELD based on your custom inputs.

The VisualSHIELDServer `servers` parameter should return a list that will contain

- user name/email of the user performing the actions. This comes handy if you have multiple users for your shiny and you want to track down which user did an operation. If you don't need to associate analysis to user you can put here whatever fixed string you like.
- a sub-list of `servers` composed of
  - `opal_server` a sub-sub-list with the credentials to access the opal server. The parameters are listed in the example above.
  - `dashin_server` a sub-sub-list with the credentials to access the dbNP server. The parameters are:
  - `url` example: https://dashin.eu/interventionstudies-test/api/
  - `username` the username for the login
  - `password` the password for the login
  - `skey` an alphanumeric string needed for the API