```
#########################################################################################################################################

#### ------------------------------------------------------------------------------------------------------------------
#####
#### ------------------------------- Ascendance: A New Paradigm in Time Management -------------------------------
#####
#### ------------------------------------------------------------------------------------------------------------------
#####
#########################################################################################################################################


#' Ascendance is the evolution of TimeProg, a predecessor program, with many more degrees of flexibility and a coherent
#' central structure. This is the program's source code, containing the functions essential for its operation, integrated
#' into a Shiny application for interactive usage.

#' To run this application for the first time you will need to uncomment the code below and run it
#' (make sure each package installs completely before installing the next, and don't forget to re-comment
#' the code when you've finished running it). Then you will need to create the following folder
#' structure somewhere in your computer: 'Ascendance > App > Setup' and replace the 'filepaths' object
#' below with the path to the first folder in that sequence (don't forget the ending slash). Additionally,
#' you will need to change the name of the R file, currently being saved to the 'paths' object below the
#' sourcing code, from 'A_Johann.R' to 'A_[your name].R'. Once all this is done, just hit the 'Run App' button
#' at the top of the screen, and you will be launched into the setup process.

#' IMPORTANT NOTES CONCERNING THE SETUP PROCESS: At the current time, there are numerous inconveniences you will
#' have to suffer to use this program. Firstly, do not click the "back" button; you will probably lose your data,
#' or other terrible things will happen. Second, do not hit 'Submit' or 'Go' buttons more than once; much of the
#' time, you will not recieve feedback that you cliked a button, but the action specified by that button will still
#' occur. Think carefully before you click 'Next'. Read all of the text in each of the modules, as they have been
#' updated to inform you about how to avoid problems.
#'
# install.packages("RColorBrewer"); install.packages("ggplot2"); install.packages("devtools")
# require(devtools); install_github("rstudio/shiny");
# install.packages("shinydashboard"); install.packages("shinyjs"); install.packages("magrittr");
# install.packages("lubridate"); install.packages("tibble")

require(RColorBrewer)
require(ggplot2)
require(shinydashboard)
require(shinyjs)
require(magrittr)
require(lubridate)
require(devtools)
require(tibble)

require(shiny)

# filepaths <- "D:/R/Ascendance/"
# filepaths <- "C:/Users/Johann/Documents/Ascendance/"
filepaths <- "/home/alan/Documents/Ascendance/"

source(paste0(filepaths, "App/GeneralFunctions.R"))
source(paste0(filepaths, "App/Setup/setup_focusAreas.R"))
source(paste0(filepaths, "App/Setup/setup_categories.R"))
source(paste0(filepaths, "App/Setup/setup_focusPeriod.R"))
source(paste0(filepaths, "App/Setup/setup_taskSchedule.R"))
source(paste0(filepaths, "App/Setup/setup_trackers.R"))
source(paste0(filepaths, "App/Setup/setup_hierarchy.R"))

#  "A_Johann.R"

path <- paste0(filepaths, "A_blackslate.R")

options(shiny.reactlog=TRUE)

#' GenFunc

nameObject <- function(name, object, Envir = .GlobalEnv) {
  nm <- name
  v <- object
  assign(nm,v, envir = Envir)
}

colfunc <- colorRampPalette(c("slategray1", "deepskyblue4"))

if (file.exists(path) == TRUE) {

  load(path, envir = .GlobalEnv)

  header <- dashboardHeader(title = "Ascendance")

  sidebar <- dashboardSidebar( sidebarMenuOutput("menu") )

  body <- dashboardBody({

    tabItems(

      tabItem(tabName = "assessment_tab",
              h2(""),
              uiOutput("uiAssess")),

      tabItem(tabName = "input_tab",
```

```
            h2(""),
            uiOutput("uiInput"))

      )

  })

  ui <- dashboardPage(header, sidebar, body)

  server <- function(input, output) {

    output$menu <- renderMenu({

      sidebarMenu(
        menuItem("Assess", tabName = "assessment_tab", icon = icon("area-chart")),
        menuItem("Manage", tabName = "main_tab", icon = icon("cog"),
                 menuSubItem("Focus areas", tabName = "Mfocus"),
                 menuSubItem("Ascendance Options", tabName = "Mao")
        ),

        menuItem("Input", tabName = "input_tab", icon = icon("archive")),
        menuItem("Predict", tabName = "prediction_tab", icon = icon("line-chart")),
        menuItem("Plan", tabName = "planning_tab", icon = icon("calendar")),
        menuItem("View reports", tabName = "report_tab", icon = icon("file"))
      )

    })

    callModule(assess, "assess", Path = path); callModule(Input, "input", Path = path)

    output$uiAssess <- renderUI({

      assessUI("assess")

    })

    output$uiInput <- renderUI({

      InputUI("input")

    })

  }

  shinyApp(ui, server)

} else {

  A <<- new.env()
  # load(path, envir = .GlobalEnv)

  ############### SIMPLE UI SETUP

  header <- dashboardHeader(title = "Ascendance")

  sidebar <- dashboardSidebar(disable = TRUE, sidebarMenuOutput("menu"))

  body <- dashboardBody(

    fluidRow(uiOutput("UI")),

    fluidRow(

      column(width = 1, actionButton("Back", "Back")),

      column(width = 1, actionButton("Next", "Next"))

    )

  )

  ui <- dashboardPage(header, sidebar, body)

  ##############

  server <- function(input, output) {

    #' ---------------------------------------| SETUP |-----------------------------------------
    #' STORAGE
    #'
    #' Store some reactive inputs to be used later

    A$rv <- list(Names = c(), Location = c()) %>% list %>% rep( . , 2)
    names(A$rv) <- c("0", "1")

    #' Create reactive values object 'tut', with attribute 'stage' equal to zero

    tut <- reactiveValues(stage = 0)

    #' Make input for 'Next' button a reactive expression that will be used by observers inside modules for observers to
fire when user
    #' clicks 'Next'
```

```
    yes <- reactive({input$Next})

    #' Store the arguments to a future 'switch' call in list form, and save as reactive object

    tut$modules <- list("0",
                    "0" = {

                      substitute(

                        callModule(setup_trackers, x, yes),

                        list(x = "0")

                      )

                    }
    )

    #' Store the arguments to a future 'switch' call in list form, and save as reactive object

    tut$moduleUIs <- list("0",
                          "0" = {

                           substitute(

                              renderUI({

                                setup_trackersInput(x)

                              }),

                              list(x = "0")

                            )

                          }
    )

    #' ---------------------------------------| DYNAMIC MODULES LIST |-----------------------------------------
    #'
    #' Use i0 rows, created after user finishes first module, to generate future modules, and insert them in the 'switch'
chain

    #' Observer fires after 'proceed()' observer in trackers module; gets new 'tut$stage' value, and

    observeEvent(tut$stage, priority = 2, {

      stage <- tut$stage %>% as.character

      if (stage == "1") {

        modIDs <- A$rv[["0"]]$Names %>% length %>% seq_len %>% as.character

        newModules <- lapply(modIDs, function(x) {

          substitute(

            callModule(setup_hierarchy, x, yes, ID = x),

            list(x = x, ID = x)

          )

        })

        newModules <- newModules %>% append(tut$modules, . )

        names(newModules) <- append(c("", "0"), modIDs)

        tut$modules <- newModules

        newModuleUIs <- lapply(modIDs, function(x) {

          substitute(

            renderUI({

              setup_hierarchyInput(x)

            }),

            list(x = x)

          )

        })

        newModuleUIs <- newModuleUIs %>% append(tut$moduleUIs, . )

        names(newModuleUIs) <- append(c("", "0"), modIDs)

        tut$moduleUIs <- newModuleUIs
```

```
      }

      # "newModules utility:" %>% print
      # tut$modules %>% print

      tut$modules[[1]] <- stage
      tut$moduleUIs[[1]] <- stage

    })

    observeEvent(tut$modules, priority = 1, {

      stage <- tut$stage %>% as.character

      "do.call utility:" %>% print
      tut$modules %>% print

      do.call(switch, tut$modules) %>% eval

      output$UI <- do.call(switch, tut$moduleUIs) %>% eval

    })

    observeEvent(input$Next, priority = 4, {

      if (tut$stage < 5) {

        tut$stage <- tut$stage + 1

      }

    })

    observeEvent(input$Back, priority = 4, {

      if (tut$stage >= 0) {

        tut$stage <- tut$stage - 1

      }

    })


  }

  shinyApp(ui, server)

}

# dropdownMenu(type = "tasks", badgeStatus = "success",
#              taskItem(value = 90, color = "green",
#                       "Documentation"
#              ),
#              taskItem(value = 17, color = "aqua",
#                       "Project X"
#              ),
#              taskItem(value = 75, color = "yellow",
#                       "Server deployment"
#              ),
#              taskItem(value = 80, color = "red",
#                       "Overall project"
#              )
#)
```