Why {golem}?

Why should you consider the {golem} package to develop your Shiny apps? For me, there are two
main reasons. First of all, I'm already familiar with package development in R, having made some
little packages that I have on my Github account, and one out on CRAN (with the complete texts of
Luxembourguish author Michel Rodange)
so using {golem} came at no additional costs. This is because a Shiny app built with {golem} is
actually an R package! This has many advantages; all the steps of documenting, testing and sharing
the app are greatly simplified.
Another reason to use {golem} is that it forces on you a certain way of working. Now this
might seem like a pretty bad thing, but I find that it is quite helpful. When you start working
on a Shiny app, you might get very quickly overwhelmed with both thinking about your server logic
and your UI. You might spend much time tinkering with getting the server functions working, while
still not having no UI to speak of, or you might work on one part of the server and then go to the UI,
then back on the server… You'll spend hours working on the app without a clear approach, and
probably waste much time because of this back and forth.
The first recommended step when building a shiny app (with or without {golem}) is a "UI first"
approach.
For this, we're going to use {shinipsum}

Lorem ipsum dolor server amet (it's Latin for "don't bother with the server logic until it's time")

The developers of {golem}, French company ThinkR
suggest an "UI" first approach. The idea is to focus on the UI, and to do so using their other
package called
{shinipsum} to randomly generate elements on the server side which you can then later replace
with your actual server logic. For instance, imagine that somewhere on your app, you want to show
a bar plot using the {ggplot2} package. Using {shinipsum}, you can generate a random bar plot
with the following line:

```
shinipsum::random_ggplot("bar")
```

and that's it! Now simply ignore this bit on the server, and continue focusing on the UI. You need
to show a random table? No problem:

```
shinipsum::random_table(ncol = 7, nrow = 10)
##    conc rate   state conc.1 rate.1 state.1 conc.2
## 1  0.02   76 treated   0.02     76 treated   0.02
## 2  0.02   47 treated   0.02     47 treated   0.02
## 3  0.06   97 treated   0.06     97 treated   0.06
## 4  0.06  107 treated   0.06    107 treated   0.06
## 5  0.11  123 treated   0.11    123 treated   0.11
## 6  0.11  139 treated   0.11    139 treated   0.11
## 7  0.22  159 treated   0.22    159 treated   0.22
## 8  0.22  152 treated   0.22    152 treated   0.22
## 9  0.56  191 treated   0.56    191 treated   0.56
## 10 0.56  201 treated   0.56    201 treated   0.56
```

Your app might now look something like this (actually, it won't because the little demo below is not a `{golem}` app, but it illustrates `{shinipsum}` well):

```
library(shiny)
library(reactable)
library(shinipsum)
library(ggiraph)

ui <- pageWithSidebar(

  headerPanel("This is a shinipsum demo"),

  sidebarPanel(
    sliderInput("rows",
                "Number of rows:",
                min = 1,
                max = 50,
                value = 5)
  ),

  mainPanel(
    reactableOutput("table"),
    girafeOutput("graph")
  )
)


server <- function(input, output) {

  output$table <- renderReactable({
    reactable(random_table(ncol = 10, nrow = input$rows))
  })

  output$graph <- renderGirafe({
    girafe(ggobj = random_ggplot("bar"))
  })
}

shinyApp(ui = ui, server = server)
```

If you have the required packages, running this on a fresh R session should start a little app.

You see that the server is only a call to `shinipsum::random_table`, and `shinipsum::random_ggplot`.
Because I want a `reactable` and an interactive plot using the `{ggiraph}` package, I have already
written the minimum amount of code on the server side to get things working. Now I can focus on my
UI and then, when I'm done, I can start replacing the random objects from `{shinipsum}` with the actual code.

Now proceeding in this way is not a requirement of `{golem}`, but it helps to structure your thoughts
and your app, and you can use this approach for any type of app. The example above, after all, is
not a `{golem}` app.

Get modular with `{golem}`

This is now where we get to some more interesting, and `{golem}` specific things. If you've been using R and Shiny for the past years, you've probably have heard a lot about functional programming.
Functional programming is a programming paradigm that encourages, and in some languages forces, the use of functions. The idea is that everything you do should be a call to a function, and functions should be chained together to achieve whatever it is you want to do; cleaning data,
visualizing data, modeling data… R has many functional tools out of the box, which can be complemented
using the `{purrr}` package.
What does all of this have to do with Shiny and `{golem}`? Well, `{golem}` forces you to write modules to build your apps, and modules are very similar to functions (they're actually functions).
They're bits of code that can be decoupled from your app, used in any other app, they can be linked together,
they can be easily documented and tested… If you are familiar with R's functional programming approach, modules should not be totally new to you. But if you've been using Shiny without module,
they'll require some getting used to.

To illustrate how a simple app can be written using modules, I have built `golemDemo`, which, as implied
by its name, is a demonstration of a `{golem}` app which I hope is simple enough for anyone to start using. The app is quite simple and does only three things:

- it allows you to choose between two datasets;
- it shows a table of the selected dataset;
- it shows a map of Luxembourg with the data points;

Each of these things is a module, which means that if I were to create another app with a map of Luxembourg, I could simply reuse it. But remember, the app is actually an R package. Here is the root of the app on my computer:

```
system2("ls", args = "-lFR ~/Documents/golemDemo", stdout = TRUE)
```

```
##  [1] "/home/cbrunos/Documents/golemDemo:"
##  [2] "total 56"
##  [3] "-rw-r--r-- 1 cbrunos users  302 Sep 19 11:28 app.R"
##  [4] "drwxr-xr-x 2 cbrunos users 4096 Jun 29 17:49 data-raw/"
##  [5] "-rw-r--r-- 1 cbrunos users  729 Sep 19 21:27 DESCRIPTION"
##  [6] "drwxr-xr-x 2 cbrunos users 4096 Sep 11 23:39 dev/"
##  [7] "-rw-r--r-- 1 cbrunos users 2723 Sep 12 15:04 Dockerfile"
##  [8] "drwxr-xr-x 3 cbrunos users 4096 Jun 28 11:33 inst/"
##  [9] "-rw-r--r-- 1 cbrunos users  483 Apr  8 21:38 LICENSE.md"
## [10] "drwxr-xr-x 2 cbrunos users 4096 Sep 19 21:27 man/"
## [11] "-rw-r--r-- 1 cbrunos users 1420 Sep 19 21:27 NAMESPACE"
## [12] "drwxr-xr-x 2 cbrunos users 4096 Sep 19 21:27 R/"
## [13] "-rw-r--r-- 1 cbrunos users 1056 Jun 28 11:38 README.Rmd"
## [14] "drwxr-xr-x 3 cbrunos users 4096 Sep 11 17:12 rsconnect/"
## [15] "drwxr-xr-x 3 cbrunos users 4096 Jun 28 11:48 tests/"
## [16] "drwxr-xr-x 2 cbrunos users 4096 Jun 28 11:48 vignettes/"
## [17] ""
## [18] "/home/cbrunos/Documents/golemDemo/data-raw:"
## [19] "total 1168"
## [20] "-rw-r--r-- 1 cbrunos users 1176106 Jun 11 09:52
communes_df.csv"
## [21] "-rw-r--r-- 1 cbrunos users      99 Jun 28 11:48 my_dataset.R"
## [22] "-rw-r--r-- 1 cbrunos users    1998 Jun 28 17:00 radars.csv"
## [23] "-rw-r--r-- 1 cbrunos users    6390 Jun 28 12:31
rettungspunkte.csv"
## [24] ""
## [25] "/home/cbrunos/Documents/golemDemo/dev:"
## [26] "total 16"
## [27] "-rw-r--r-- 1 cbrunos users 1935 Jun 28 11:33 01_start.R"
## [28] "-rw-r--r-- 1 cbrunos users 2011 Sep 11 23:39 02_dev.R"
## [29] "-rw-r--r-- 1 cbrunos users 1012 Jun 28 11:33 03_deploy.R"
## [30] "-rw-r--r-- 1 cbrunos users  318 Jun 28 11:33 run_dev.R"
## [31] ""
## [32] "/home/cbrunos/Documents/golemDemo/inst:"
## [33] "total 8"
## [34] "drwxr-xr-x 3 cbrunos users 4096 Jun 28 11:33 app/"
## [35] "-rw-r--r-- 1 cbrunos users  140 Jun 28 11:38 golem-config.yml"
## [36] ""
## [37] "/home/cbrunos/Documents/golemDemo/inst/app:"
## [38] "total 4"
## [39] "drwxr-xr-x 2 cbrunos users 4096 Jun 28 11:48 www/"
## [40] ""
## [41] "/home/cbrunos/Documents/golemDemo/inst/app/www:"
## [42] "total 12"
## [43] "-rw-r--r-- 1 cbrunos users    0 Jun 28 11:48 custom.css"
## [44] "-rw-r--r-- 1 cbrunos users 3774 Jun 28 11:33 favicon.ico"
## [45] "-rw-r--r-- 1 cbrunos users  100 Jun 28 11:48 handlers.js"
## [46] "-rw-r--r-- 1 cbrunos users   40 Jun 28 11:48 script.js"
## [47] ""
## [48] "/home/cbrunos/Documents/golemDemo/man:"
## [49] "total 8"
## [50] "-rw-r--r-- 1 cbrunos users 261 Sep 19 21:27 pipe.Rd"
```

```
## [51] "-rw-r--r-- 1 cbrunos users 291 Jun 28 11:33 run_app.Rd"
## [52] ""
## [53] "/home/cbrunos/Documents/golemDemo/R:"
## [54] "total 48"
## [55] "-rw-r--r-- 1 cbrunos users  783 Jun 28 11:33 app_config.R"
## [56] "-rw-r--r-- 1 cbrunos users  654 Jun 29 18:34 app_server.R"
## [57] "-rw-r--r-- 1 cbrunos users 1790 Sep 12 15:00 app_ui.R"
## [58] "-rw-r--r-- 1 cbrunos users    0 Jun 28 11:48 fct_helpers.R"
## [59] "-rw-rw-r-- 1 cbrunos users  997 Jun 28 11:38
golem_utils_server.R"
## [60] "-rw-rw-r-- 1 cbrunos users 5849 Jun 28 11:38 golem_utils_ui.R"
## [61] "-rw-r--r-- 1 cbrunos users  549 Jun 28 11:48
mod_filter_data.R"
## [62] "-rw-r--r-- 1 cbrunos users 3118 Sep 19 11:16 mod_load_data.R"
## [63] "-rw-r--r-- 1 cbrunos users 2088 Jun 29 18:30 mod_map_data.R"
## [64] "-rw-r--r-- 1 cbrunos users  910 Jun 29 18:17 mod_table_data.R"
## [65] "-rw-r--r-- 1 cbrunos users  337 Jun 28 11:33 run_app.R"
## [66] "-rw-r--r-- 1 cbrunos users    0 Jun 28 11:48 utils_helpers.R"
## [67] "-rw-r--r-- 1 cbrunos users  207 Sep 19 21:27 utils-pipe.R"
## [68] ""
## [69] "/home/cbrunos/Documents/golemDemo/rsconnect:"
## [70] "total 4"
## [71] "drwxr-xr-x 3 cbrunos users 4096 Sep 11 17:12 shinyapps.io/"
## [72] ""
## [73] "/home/cbrunos/Documents/golemDemo/rsconnect/shinyapps.io:"
## [74] "total 4"
## [75] "drwxr-xr-x 2 cbrunos users 4096 Sep 11 17:12 brodriguesco/"
## [76] ""
## [77] "/home/cbrunos/Documents/golemDemo/rsconnect/shinyapps.
io/brodriguesco:"
## [78] "total 4"
## [79] "-rw-r--r-- 1 cbrunos users 219 Sep 19 21:30 golemdemo.dcf"
## [80] ""
## [81] "/home/cbrunos/Documents/golemDemo/tests:"
## [82] "total 8"
## [83] "drwxr-xr-x 2 cbrunos users 4096 Jun 28 11:48 testthat/"
## [84] "-rw-r--r-- 1 cbrunos users   62 Jun 28 11:48 testthat.R"
## [85] ""
## [86] "/home/cbrunos/Documents/golemDemo/tests/testthat:"
## [87] "total 4"
## [88] "-rw-r--r-- 1 cbrunos users 64 Jun 28 11:48 test-app.R"
## [89] ""
## [90] "/home/cbrunos/Documents/golemDemo/vignettes:"
## [91] "total 4"
## [92] "-rw-r--r-- 1 cbrunos users 298 Jun 28 11:48 golemDemo.Rmd"
```

The first 16 lines show the root of the folder, and then we see what's inside each subfolder, starting with `data-raw/`, then `dev/` etc (this is done via a call to the `ls -lFR` Linux command, invoked here with R's `system2()` function).

If you've already developed a package in the past, you'll recognize the structure. What's important

here is the `dev/` folder, which is `{golem}` specific. This folder contains for files, `01_start.R`, `02_dev.R`, `03_deploy.R` and `run_dev.R`. These files are the ones that will help
you develop your shiny app and you should follow the instructions contained in each of them. Let's
take a look at `01_start.R`:

```
system2("cat", args = "~/Documents/golemDemo/dev/01_start.R", stdout =
TRUE)
##  [1] "# Building a Prod-Ready, Robust Shiny Application."
##  [2] "# "
##  [3] "# README: each step of the dev files is optional, and you
don't have to "
##  [4] "# fill every dev scripts before getting started. "
##  [5] "# 01_start.R should be filled at start. "
##  [6] "# 02_dev.R should be used to keep track of your development
during the project."
##  [7] "# 03_deploy.R should be used once you need to deploy your
app."
##  [8] "# "
##  [9] "# "
## [10] "########################################"
## [11] "#### CURRENT FILE: ON START SCRIPT #####"
## [12] "########################################"
## [13] ""
## [14] "## Fill the DESCRIPTION ----"
## [15] "## Add meta data about your application"
## [16] "golem::fill_desc("
## [17] "  pkg_name = \"golemDemo\", # The Name of the package
containing the App "
## [18] "  pkg_title = \"PKG_TITLE\", # The Title of the package
containing the App "
## [19] "  pkg_description = \"PKG_DESC.\", # The Description of the
package containing the App "
## [20] "  author_first_name = \"AUTHOR_FIRST\", # Your First Name"
## [21] "  author_last_name = \"AUTHOR_LAST\", # Your Last Name"
## [22] "  author_email = \"AUTHOR@MAIL.COM\", # Your Email"
## [23] "  repo_url = NULL # The URL of the GitHub Repo (optional) "
## [24] ")     "
## [25] ""
## [26] "## Set {golem} options ----"
## [27] "golem::set_golem_options()"
## [28] ""
## [29] "## Create Common Files ----"
## [30] "## See ?usethis for more information"
## [31] "usethis::use_mit_license( name = \"Golem User\" )  # You can
set another license here"
## [32] "usethis::use_readme_rmd( open = FALSE )"
## [33] "usethis::use_code_of_conduct()"
## [34] "usethis::use_lifecycle_badge( \"Experimental\" )"
## [35] "usethis::use_news_md( open = FALSE )"
## [36] ""
```

```
## [37] "## Use git ----"
## [38] "usethis::use_git()"
## [39] ""
## [40] "## Init Testing Infrastructure ----"
## [41] "## Create a template for tests"
## [42] "golem::use_recommended_tests()"
## [43] ""
## [44] "## Use Recommended Packages ----"
## [45] "golem::use_recommended_deps()"
## [46] ""
## [47] "## Favicon ----"
## [48] "# If you want to change the favicon (default is golem's one)"
## [49] "golem::remove_favicon()"
## [50] "golem::use_favicon() # path = \"path/to/ico\". Can be an
online file. "
## [51] ""
## [52] "## Add helper functions ----"
## [53] "golem::use_utils_ui()"
## [54] "golem::use_utils_server()"
## [55] ""
## [56] "# You're now set! ----"
## [57] ""
## [58] "# go to dev/02_dev.R"
## [59] "rstudioapi::navigateToFile( \"dev/02_dev.R\" )"
## [60] ""
```

This script is a series of calls to `{usethis}` functions; you can remove whatever you don't need
and adapt the others that you need. As you can see, I did not change much here. Execute it line by
line when you're done editing it. Once you're done, you can go to `02_dev.R` and this is probably
the script that you'll change the most:

```
system2("cat", args = "~/Documents/golemDemo/dev/02_dev.R", stdout =
TRUE)
##  [1] "# Building a Prod-Ready, Robust Shiny Application."
##  [2] "# "
##  [3] "# README: each step of the dev files is optional, and you
don't have to "
##  [4] "# fill every dev scripts before getting started. "
##  [5] "# 01_start.R should be filled at start. "
##  [6] "# 02_dev.R should be used to keep track of your development
during the project."
##  [7] "# 03_deploy.R should be used once you need to deploy your
app."
##  [8] "# "
##  [9] "# "
## [10] "####################################"
## [11] "#### CURRENT FILE: DEV SCRIPT #####"
## [12] "####################################"
## [13] ""
```

```
## [14] "# Engineering"
## [15] ""
## [16] "## Dependencies ----"
## [17] "## Add one line by package you want to add as dependency"
## [18] "usethis::use_package( \"shiny\" )"
## [19] "usethis::use_package( \"shinydashboard\" )"
## [20] "usethis::use_package(\"data.table\") "
## [21] "usethis::use_package(\"DT\")"
## [22] "usethis::use_package(\"dplyr\")"
## [23] "usethis::use_package(\"rlang\")"
## [24] "usethis::use_package(\"ggiraph\")"
## [25] "usethis::use_package(\"ggplot2\")"
## [26] "usethis::use_package(\"htmlwidgets\")"
## [27] "usethis::use_package(\"dplyr\")"
## [28] "usethis::use_package(\"colorspace\")"
## [29] "usethis::use_package(\"shinycssloaders\")"
## [30] "usethis::use_package(\"lubridate\")"
## [31] ""
## [32] "## Add modules ----"
## [33] "## Create a module infrastructure in R/"
## [34] "golem::add_module( name = \"name_of_module1\" ) # Name of the
module"
## [35] "golem::add_module( name = \"name_of_module2\" ) # Name of the
module"
## [36] ""
## [37] "## Add helper functions ----"
## [38] "## Creates ftc_* and utils_*"
## [39] "golem::add_fct( \"helpers\" ) "
## [40] "golem::add_utils( \"helpers\" )"
## [41] ""
## [42] "## External resources"
## [43] "## Creates .js and .css files at inst/app/www"
## [44] "golem::add_js_file( \"script\" )"
## [45] "golem::add_js_handler( \"handlers\" )"
## [46] "golem::add_css_file( \"custom\" )"
## [47] ""
## [48] "## Add internal datasets ----"
## [49] "## If you have data in your package"
## [50] "usethis::use_data_raw( name = \"my_dataset\", open = FALSE ) "
## [51] ""
## [52] "## Tests ----"
## [53] "## Add one line by test you want to create"
## [54] "usethis::use_test( \"app\" )"
## [55] ""
## [56] "# Documentation"
## [57] ""
## [58] "## Vignette ----"
## [59] "usethis::use_vignette(\"golemDemo\")"
## [60] "devtools::build_vignettes()"
## [61] ""
## [62] "## Code coverage ----"
## [63] "## (You'll need GitHub there)"
```

```
## [64] "usethis::use_github()"
## [65] "usethis::use_travis()"
## [66] "usethis::use_appveyor()"
## [67] ""
## [68] "# You're now set! ----"
## [69] "# go to dev/03_deploy.R"
## [70] "rstudioapi::navigateToFile(\"dev/03_deploy.R\")"
## [71] ""
```

This is where you will list the dependencies of your package (lines 18 to 30) as well as the modules (lines 34 to 35). I have mostly used this file for the dependencies, as I already had the modules from another app, so I didn't bother listing them here. But if I would have started
from scratch, I would changed the line:

```
golem::add_module( name = \"name_of_module1\" ) # Name of the module
```

to something like:

```
golem::add_module( name = \"import_data\" ) # Name of the module
```

and executing it would have generated the needed files to start creating the module at the right spot. Let's go see how such a module looks like (I'm skipping the third script for now, as it is only useful once you want to deploy).

You can find the modules in the `R/` folder. Let's take a look at the module that allows the user to load the data:

```
system2("cat", args = "~/Documents/golemDemo/R/mod_load_data.R", stdout
= TRUE)
##   [1] "#' load_data UI Function"
##   [2] "#'"
##   [3] "#' @description A shiny Module."
##   [4] "#'"
##   [5] "#' @param id,input,output,session Internal parameters for
{shiny}."
##   [6] "#'"
##   [7] "#' @noRd "
##   [8] "#'"
##   [9] "#' @importFrom shiny NS tagList "
##  [10] "#' @importFrom data.table fread"
##  [11] "#' @importFrom DT renderDataTable dataTableOutput"
##  [12] "#' @importFrom dplyr filter"
##  [13] "#' @importFrom rlang quo `!!` as_name"
##  [14] "mod_load_data_ui <- function(id){"
##  [15] "  ns <- NS(id)"
##  [16] "  tagList("
##  [17] "    box(title = \"Select dataset\","
##  [18] "        radioButtons(ns(\"select_dataset\"),"
##  [19] "                     label = \"Select dataset\","
##  [20] "                     choices = c(\"Rescue points\",
\"Radars\"),"
##  [21] "                     selected = c(\"Rescue points\")),"
##  [22] "        conditionalPanel("
```

```
##  [23] "          condition = paste0('input[\\'',
ns('select_dataset'), \"\\'] == \\'Rescue points\\'\"),"
##  [24] "          selectInput(ns(\"selector_place\"), \"Place\","
##  [25] "                      choices = c(\"test\"),"
##  [26] "                      #choices =
c(unique(output$dataset$place)),"
##  [27] "                      selected = c(\"Luxembourg, Ville
(G)\"),"
##  [28] "                      multiple = TRUE)),"
##  [29] "        conditionalPanel("
##  [30] "          condition = paste0('input[\\'',
ns('select_dataset'), \"\\'] == \\'Radars\\'\"),"
##  [31] "          selectInput(ns(\"selector_radar\"), \"Radar\","
##  [32] "                      choices = c(\"test\"),"
##  [33] "                      #choices = c(\"huhu\"),"
##  [34] "                      selected = c(\"National road\"),"
##  [35] "                      multiple = TRUE)),"
##  [36] "        width = NULL),"
##  [37] "  )"
##  [38] "}"
##  [39] ""
##  [40] "#' load_data Server Function"
##  [41] "#'"
##  [42] "#' @noRd "
##  [43] "mod_load_data_server <- function(input, output, session){"
##  [44] "  ns <- session$ns"
##  [45] " "
##  [46] "  "
##  [47] "  read_dataset <- reactive({"
##  [48] "    if(input$select_dataset == \"Rescue points\") {"
##  [49] ""
##  [50] "      dataset <- fread(\"data-raw/rettungspunkte.csv\")"
##  [51] "      variable <- quo(place)"
##  [52] "      filter_values <- unique(dataset[, place])"
##  [53] "    } else {"
##  [54] "      dataset <- fread(\"data-raw/radars.csv\")"
##  [55] "      variable <- quo(type_road)"
##  [56] "      filter_values <- unique(dataset[, type_road])"
##  [57] "    }"
##  [58] "    cat(\"reading data\\n\")"
##  [59] "    list(dataset = dataset,"
##  [60] "         variable = variable,"
##  [61] "         filter_values = filter_values)"
##  [62] "  })"
##  [63] ""
##  [64] ""
##  [65] "  observe({"
##  [66] "    updateSelectInput(session, \"selector_place\", label =
\"Select place:\", choices = read_dataset()$filter_values,"
##  [67] "                      selected = \"Luxembourg, Ville (G)\")"
##  [68] "  })"
##  [69] ""
```

```
##  [70] "  observe({"
##  [71] "    updateSelectInput(session, \"selector_radar\", label =
\"Select type of road:\", choices = read_dataset()$filter_values,"
##  [72] "                      selected = \"National road\")"
##  [73] "  })"
##  [74] ""
##  [75] "  result <- reactive({"
##  [76] "    return_dataset <- read_dataset()$dataset"
##  [77] ""
##  [78] "    if(\"place\" %in% colnames(return_dataset)){"
##  [79] "      return_dataset <- return_dataset %>%"
##  [80] "        filter((!!read_dataset()$variable) %in%
input$selector_place)"
##  [81] ""
##  [82] "      result <- list("
##  [83] "        return_dataset = return_dataset,"
##  [84] "        variable = quo(place)"
##  [85] "      )"
##  [86] "    } else {"
##  [87] "      return_dataset <- return_dataset %>%"
##  [88] "        filter((!!read_dataset()$variable) %in%
input$selector_radar)"
##  [89] ""
##  [90] "      result <- list("
##  [91] "        return_dataset = return_dataset,"
##  [92] "        variable = quo(type_road)"
##  [93] "      )"
##  [94] "    }"
##  [95] "  })"
##  [96] ""
##  [97] "  result"
##  [98] "}"
##  [99] "    "
## [100] "## To be copied in the UI"
## [101] "# mod_load_data_ui(\"load_data_ui_1\")"
## [102] "    "
## [103] "## To be copied in the server"
## [104] "# callModule(mod_load_data_server, \"load_data_ui_1\")"
## [105] " "
```

This scripts looks like a mini Shiny app; there's a UI defined at the top of the script, and then a server defined at the bottom (I'm not describing what the module does here, I'll do that in the video). What's important here, is that this is a module and as such it can be reused in any app, by simply copying the right lines of code at the right spot. See lines 100 to 104 for this, which tells you exactly where to copy the lines to use this module. All the modules will look the same, and have this little explanation at the bottom to tell you where you need to copy the lines to use the modules. While building each module, you can use `{shinipsum}` instead of
having to bother about the server logic, just to get things going, as explained above.

Now, finally, let's take a look at the actual UI of the app:

```
system2("cat", args = "~/Documents/golemDemo/R/app_ui.R", stdout =
TRUE)
##  [1] "#' The application User-Interface"
##  [2] "#' "
##  [3] "#' @param request Internal parameter for `{shiny}`. "
##  [4] "#'     DO NOT REMOVE."
##  [5] "#' @import shiny"
##  [6] "#' @import shinydashboard"
##  [7] "#' @noRd"
##  [8] "app_ui <- function(request) {"
##  [9] "  tagList("
## [10] "                                          # Leave this function
for adding external resources"
## [11] "    golem_add_external_resources(),"
## [12] "                                          # List the first level
UI elements here"
## [13] "    dashboardPage("
## [14] "      dashboardHeader(title = \"Prototype: dashboard
ecoles\"),"
## [15] "      dashboardSidebar("
## [16] "        sidebarMenu("
## [17] "          menuItem(\"Carte\", tabName = \"Carte\", icon =
icon(\"map\")),"
## [18] "          menuItem(\"Tab 2\", tabName = \"tab_2\", icon =
icon(\"chart-line\"))"
## [19] "        )"
## [20] "      ),"
## [21] "      dashboardBody("
## [22] "        tabItems("
## [23] "          tabItem(tabName = \"Carte\","
## [24] "                  fluidRow("
## [25] "                    column("
## [26] "                      width = 4,"
## [27] "                      mod_load_data_ui(\"load_data_ui_1\"),"
## [28] "                      mod_table_data_ui(\"table_data_ui_1\")"
## [29] "                    ),"
## [30] "                    column("
## [31] "                      width = 6, offset = 2,"
## [32] "                      mod_map_data_ui(\"map_data_ui_1\")"
## [33] "                    )"
## [34] "                  ))"
## [35] "        )"
## [36] "      )"
## [37] "    )"
## [38] "  )"
## [39] "}"
## [40] ""
## [41] "#' Add external Resources to the Application"
## [42] "#' "
## [43] "#' This function is internally used to add external "
## [44] "#' resources inside the Shiny application. "
## [45] "#' "
```

```
## [46] "#' @import shiny"
## [47] "#' @importFrom golem add_resource_path activate_js favicon
bundle_resources"
## [48] "#' @noRd"
## [49] "golem_add_external_resources <- function(){"
## [50] "  "
## [51] "  add_resource_path("
## [52] "    'www', app_sys('app/www')"
## [53] "  )"
## [54] " "
## [55] "  tags$head("
## [56] "    favicon(),"
## [57] "    bundle_resources("
## [58] "      path = app_sys('app/www'),"
## [59] "      app_title = 'golemDemo'"
## [60] "    )"
## [61] "    # Add here other external resources"
## [62] "    # for example, you can add shinyalert::useShinyalert() "
## [63] "  )"
## [64] "}"
## [65] ""
```

this is the “global” UI of the app. This looks like any other Shiny UI, but instead of having many many lines of code, there’s basically only calls to the UIs of each modules (see lines 27 and 28).
And that’s it! It keeps your code quite small and much easier to reason about. You’ll find something even simpler for the server:

```
system2("cat", args = "~/Documents/golemDemo/R/app_server.R", stdout =
TRUE)
##  [1] "#' The application server-side"
##  [2] "#' "
##  [3] "#' @param input,output,session Internal parameters for
{shiny}. "
##  [4] "#'     DO NOT REMOVE."
##  [5] "#' @import shiny"
##  [6] "#' @noRd"
##  [7] "app_server <- function( input, output, session ) {"
##  [8] "  # List the first level callModules here"
##  [9] ""
## [10] "  result <- callModule(mod_load_data_server,
\"load_data_ui_1\")"
## [11] ""
## [12] "  callModule(mod_table_data_server, \"table_data_ui_1\",
result)"
## [13] "  "
## [14] ""
## [15] "  selected_lines <- reactive({"
## [16] "    if(is.null(input$`table_data_ui_1-
dataset_rows_selected`)){"
## [17] "      return(TRUE)"
## [18] "    } else {"
```

```
## [19] "      as.numeric(input$`table_data_ui_1-
dataset_rows_selected`)"
## [20] "    }"
## [21] "  })"
## [22] ""
## [23] "  callModule(mod_map_data_server, \"map_data_ui_1\", result,
selected_lines)"
## [24] ""
## [25] "}"
```

Line 10 calls the server side of the “load data” module, and saves the result (a data frame) into a variable called `result`. This result is then passed as an argument to the server side of table data module, which simply shows a table of the data. From lines 15 to 21, I define a variable called `selected-lines` in which the lines that the user selects in the data table are saved. This gave me some headaches, because I needed to find the right syntax. I was able to find
it thanks to a Stackoverflow post that I have now lost since then… but the idea is that
the indices of the selected rows are saved into a variable called `dataset_rows_selected` and this variable name
must be appended to the name of the UI of the table where the table is. If no row is selected, then
this object should be `TRUE`; why? Because if you filter a data frame with a condition that simply evaluates always to `TRUE`, you get all the rows back, and thus, all of the data frame. If you start selecting rows, say, rows number 2, 8 and 12, then `dataset_rows_selected` will be equal to `c(2, 8, 12)`, and the filter will return these rows.

Finally, I call the module that returns a map of Luxembourg, and pass both the data frame, saved in
the `result` variable, and the `selected_lines` objects as arguments. And that’s how you make modules
communicate and share data with each other, just like you would chain functions together.
I won’t go through each module, but there’s several other interesting tricks that I’ll discuss during the video; for instance, I’m quite happy with the module that loads the data; the user can choose between two different dataset, and the select input will update with the right columns. This
also wasn’t so easy to do, but it’ll be easier to explain during a video, so stay tuned!