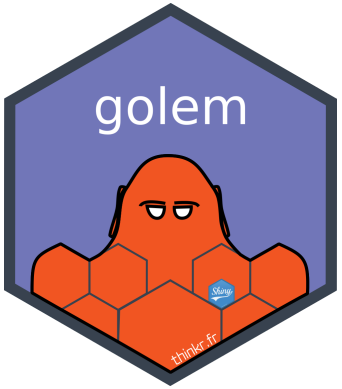


Production-grade Shiny Apps with {golem}

Colin Fay - ThinkR

ThinkR x RStudio Roadshow, Paris

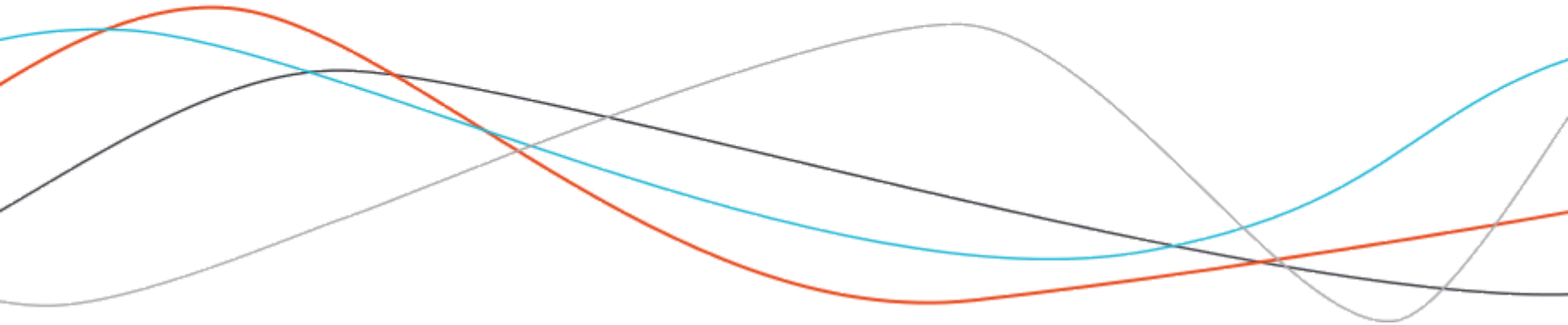
$ whoami

Colin FAY

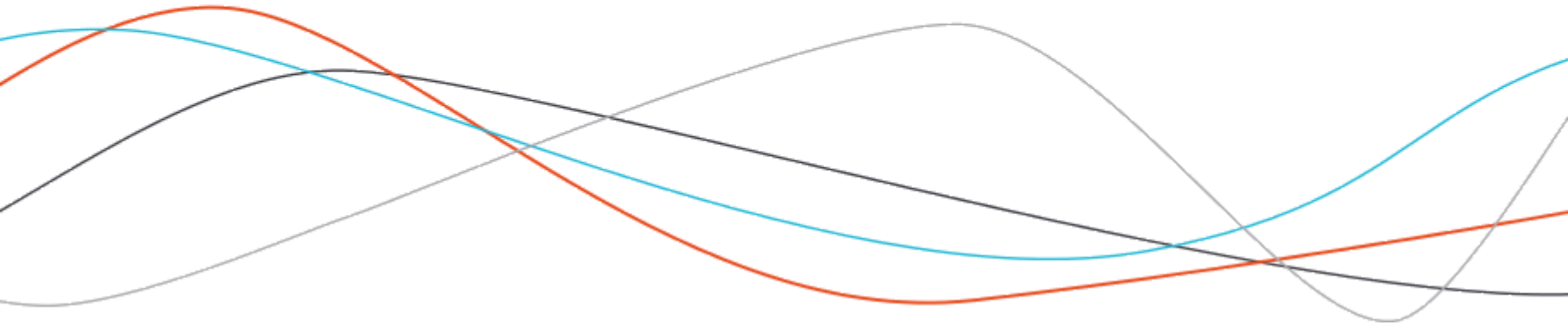
Data Scientist & R-Hacker at ThinkR

- http://thinkr.fr
- http://rtask.thinkr.fr
- http://twitter.com/thinkr_fr
- http://twitter.com/_colinfay
- http://github.com/Thinkr-open
- http://github.com/colinfay

>_ Before we start: access the RStudio Server for this workshop



About {golem}



{golem} ?

`{golem}` is an **R package**, its goal is to provide a framework for building production-ready Shiny Applications.

The framework provided by `{golem}` is relatively strict, but allows to abstract away the technical points and pure engineering steps.

Install {golem}

```
# install.packages("remotes")
remotes::install_github("Thinkr-open/golem")
```

> Notes: there are a thousand ways to create a Shiny App, but very few ways to create a production-grade Shiny App. `{golem}` provides a framework to create what we believe to be a production ready Shiny App.

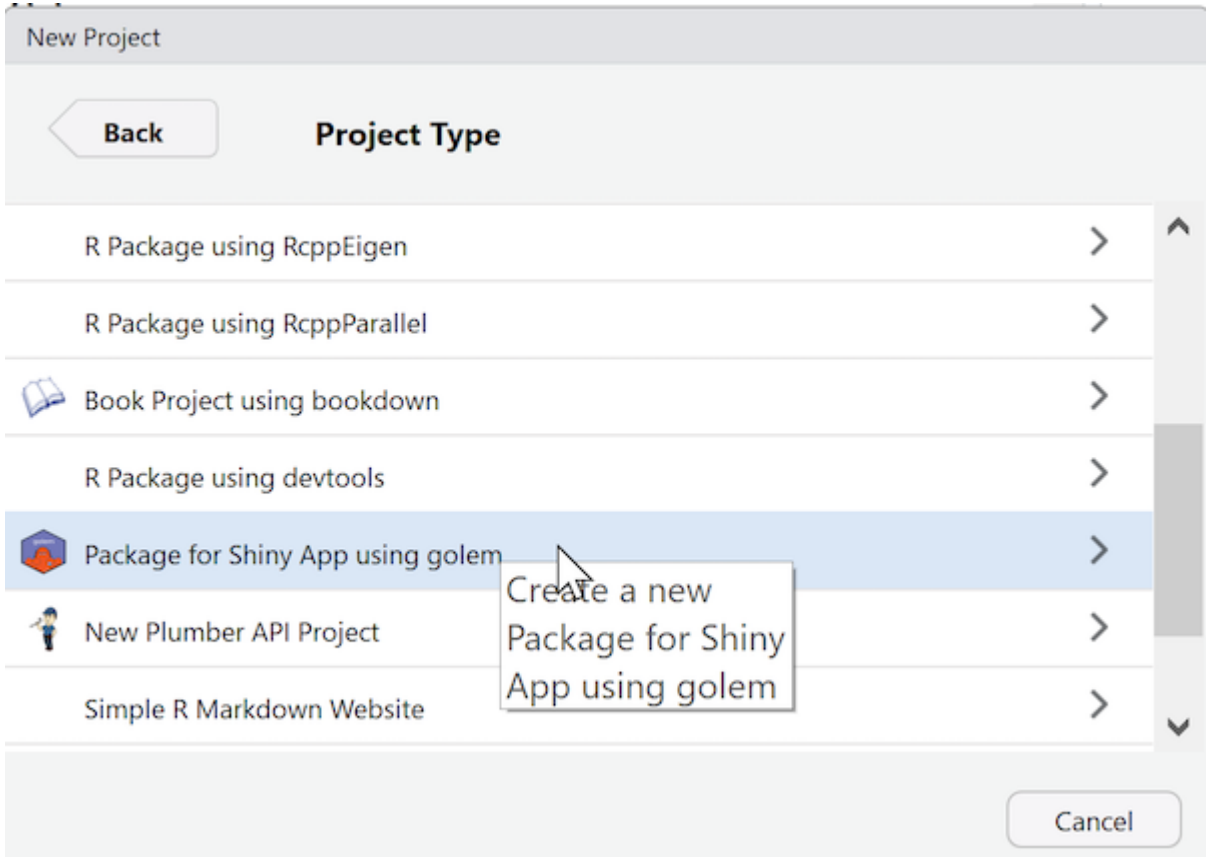
Shiny App As a Package

What's a "prod-ready" Shiny App?

- Comes with meta data (`DESCRIPTION`)
- Divided in functions (`R/`)
- Tested (`tests/`)
- With dependencies (`NAMESPACE`)
- Documented (`man/` & `vignettes`)

So, a 📦

Create a {golem}



Your turn:

>_ Create your first {golem}.

Test the app locally

To launch the app locally, run the `dev/run_dev.R` script.

>_ Open run_dev.R, run the script

>_ Add an header in app_ui

>_ Relaunch run_dev.R

Deploy on Connect

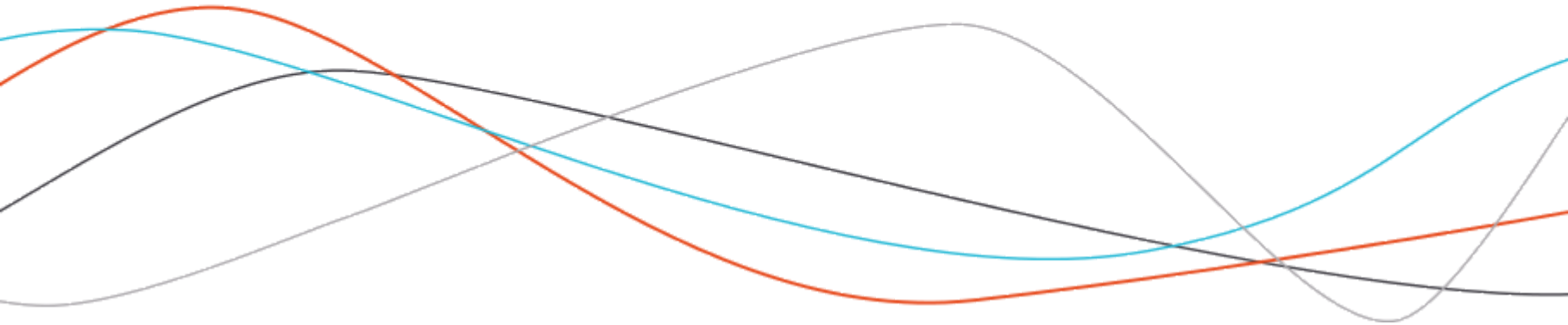
```
golem::add_rconnect_file()
usethis::use_build_ignore("app.R")
```

> Will create an `app.R` in the package folder, with in it:

```
# To deploy, run: rsconnect::deployApp()

pkgload::load_all()
options( "golem.app.prod" = TRUE)
shiny::shinyApp(ui = app_ui(), server = app_server)
```

> Now we can deploy to Connect

Understand {golem}



```
DESCRIPTION
  ├--dev/
    ├--01_start.R
    ├--02_dev.R
    ├--03_deploy.R
    ├--run_dev.R
  ├--inst/
    ├--app
      ├--server.R
      ├--ui.R
      ├--www/
        ├--favicon.ico
  ├--man/
    ├--run_app.Rd
NAMESPACE
monapp.Rproj
  ├--R/
    ├--app_server.R
    ├--app_ui.R
    ├--onload.R
    ├--run_app.R
```

- DESCRIPTION & NAMESPACE: Meta data about the package.
- dev/: dev tools.
- inst/app: We'll add external files in the `www/` folder. Leave `ui.R` & `server.R` as is.
- man: app documentation, will be automatically generated.
- monapp.Rproj : RStudio project.
- R/app_server.R, app_ui.R : these files will be filled with our modules.
- R/run_app.R & onload.R : functions that launch & configure the app.

01_start.R

Let's start with filling the `DESCRIPTION`:

```
golem::fill_desc(
  pkg_name = "nasapp",
  pkg_title = "Visualisation en Direct de l'ISS",
  pkg_description = "Visualisation en Direct la position de l'ISS",
  author_first_name = "colin",
  author_last_name = "fay" ,
  author_email = "colin@thinkr.fr",
  repo_url = NULL
)
```

01_start.R

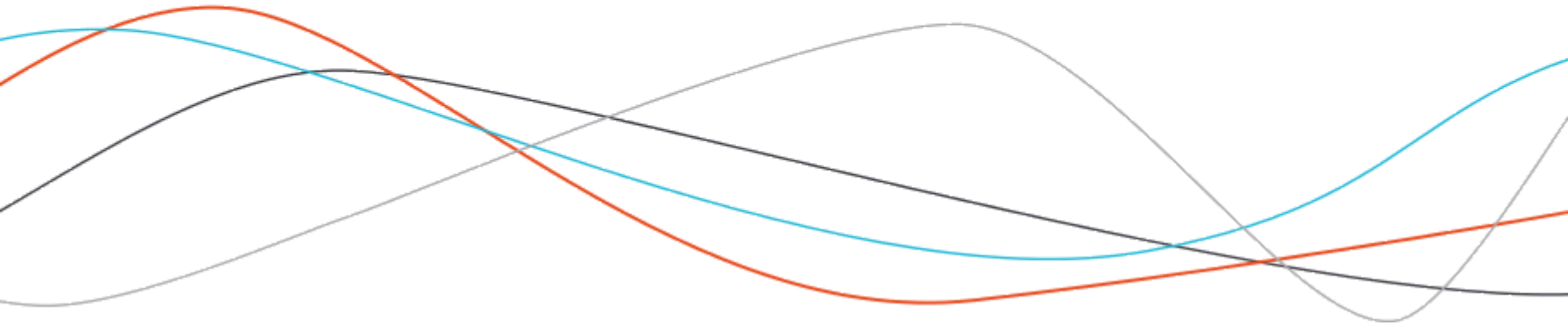
```
usethis::use_mit_license( name = "Colin Fay" )
usethis::use_readme_rmd()
usethis::use_code_of_conduct()
usethis::use_lifecycle_badge( "Experimental" )
usethis::use_news_md()

usethis::use_data_raw()

golem::use_recommended_tests()

golem::use_recommended_dep()

golem::use_utils_ui()
golem::use_utils_server()

golem::use_favicon( path = "path/to/favicon" )
```

Create modules

Ready for modules now!

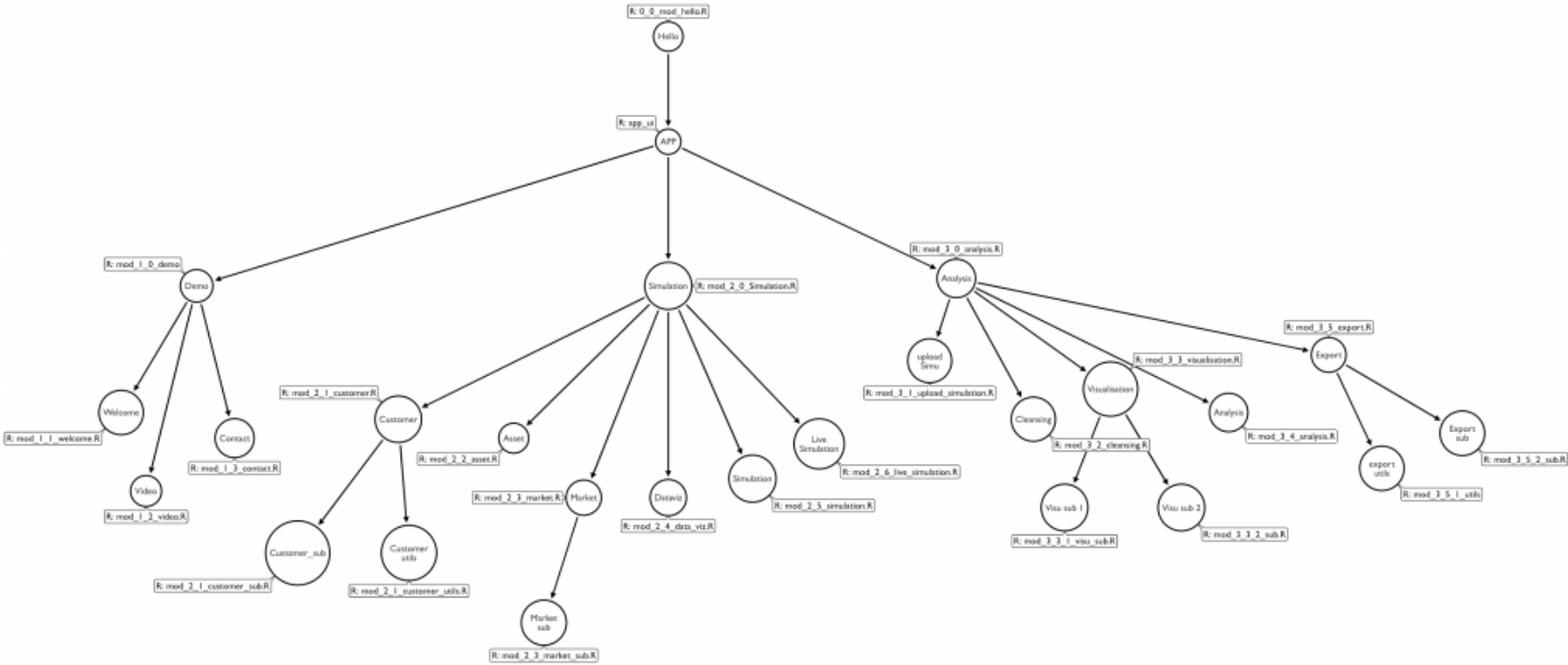
```
golem::add_module("premier_element")
```

About modules



What's a module?

- A module is a piece of Shiny App, "self contained", which will be included in a bigger app.
- It's used to split your app in smaller pieces.
- It makes handling big apps easier.
- It can be reused.


R: 0_0_mod_hello.R
Hello
R: app_ui
APP
R: mod_1_0_demo
Demo
Welcome
R: mod_1_1_welcome.R
Video
R: mod_1_2_video.R
Contact
R: mod_1_3_contact.R
Simulation
R: mod_2_0_Simulation.R
R: mod_2_1_customer.R
Customer
Customer_sub
R: mod_2_1_customer_sub.R
Customer utils
R: mod_2_1_customer_utils.R
Asset
R: mod_2_2_asset.R
Market
R: mod_2_3_market.R
Market sub
R: mod_2_3_market_sub.R
Dataviz
R: mod_2_4_data_viz.R
Simulation
R: mod_2_5_simulation.R
Live Simulation
R: mod_2_6_live_simulation.R
R: mod_3_0_analysis.R
Analysis
upload Simu
R: mod_3_1_upload_simulation.R
Cleansing
R: mod_3_2_cleansing.R
Visualisation
R: mod_3_3_visualisation.R
Visu sub 1
R: mod_3_3_1_visu_sub.R
Visu sub 2
R: mod_3_3_2_sub.R
Analysis
R: mod_3_4_analysis.R
R: mod_3_5_export.R
Export
export utils
R: mod_3_5_1_utils
Export sub
R: mod_3_5_2_sub.R

First module

```
mod_premier_elementui <- function(id){
  ns <- NS(id)
  tagList(
  # On met ici les inputs
  # All the "id" should be put inside `ns()`
  )
}

mod_premier_element <- function(input, output, session){
  ns <- session$ns
  # We'll receive the input there, and use them without ns()
}
```

Put the module in the app

```
app_ui <- function() {

  fluidPage(

    titlePanel("Old Faithful Geyser Data"),

     mod_premier_elementui("premier_elementui_1")

  )

}

app_server <- function(input, output,session) {

   callModule(mod_premier_element, "premier_elementui_1")

}
```

Example

```
mod_premier_element_ui <- function(id){
  ns <- NS(id)
  tagList(
    selectInput(
      ns("table"),
      "Which data.frame?",
      c("iris", "mtcars", "airquality")
    ),
    tableOutput(ns("out_table"))
  )
}
```

Example

```
mod_premier_element_server <- function(input, output, session) {

  output$out_table <- renderTable({
    head(get(input$table))
  })

}
```

```
shinyApp(app_ui(), app_server)
```

Summary

golem & modules

- `app_ui` : the function that defines the UI, which will be filled with ui modules: `mod_***ui( "***ui_1" )` .
- `app_server` : the server logic, defining how the app interacts with the UI. Here, you'll find a series of `callModule(mod_***, "mod_***ui_1")`
- Every module is made of a combination of UI & Server

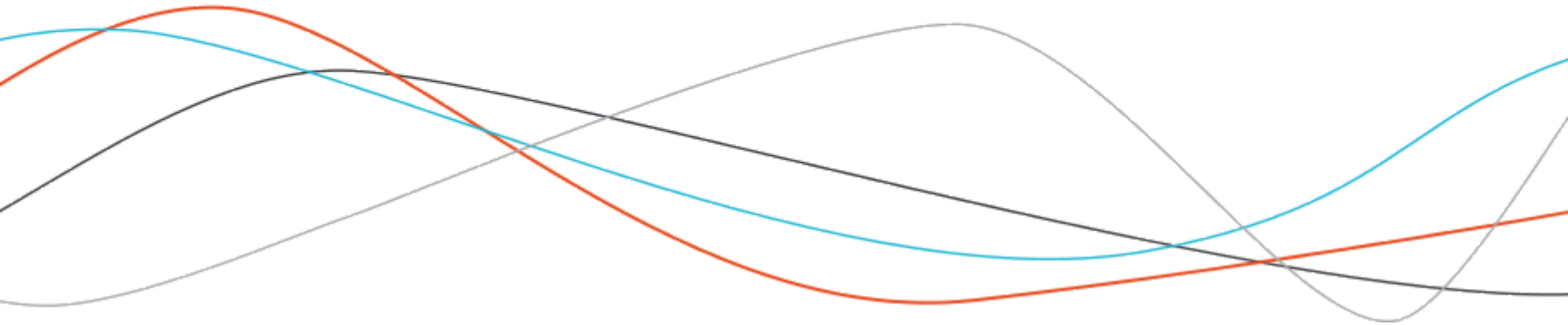
Your turn

>_ Create a first module in your golem

{golem} - workflow

- Launch the project
- Fill the DESC in `dev/01_start.R`, and run the functions.
- Launch `dev/run_dev.R` to check that everything is OK
- Close `dev/01_start.R`
- In `dev/02_dev.R`, create a module with `golem::add_module("plop")`.
- Copy and paste `module_plop_ui("plop_ui_1")` in `R/app_ui.R`
- Copy and paste `callModule(module_plop_server, "plop_ui_1")` in `R/app_server.R`
- Complete the module
- Launch `dev/run_dev.R` regularly to check that everything is fine
- Create a second module, and a third, and a fourth...

Test, deps



02_dev.R

Dependencies

```
usethis::use_package("pkg") # To call each time you need a new package
```

tests

```
usethis::use_test("app")
```

What do we test?

In our golem, there are two kinds of functions:

- back-end functions: which are "classical" functions, they should be tested as regular functions, as they don't rely on the app being run.
- front-end functions: they generate HTML, and `{golem}` has a function to test that.

```
mod_premier_elementui <- function(id){
  ns <- NS(id)
  tagList(
    sidebarLayout(
      sidebarPanel(
        sliderInput(ns("bins"),
                    "Nombre de bins:",
                    min = 1,
                    max = 50,
                    value = 30)
      ),
      mainPanel(
        plotOutput("distPlot")
      )
```

What do we test?

```
mod_premier_elementui("plop")
```

```
<div class="row">
  <div class="col-sm-4">
    <form class="well">
      <div class="form-group shiny-input-container">
        <label class="control-label" for="plop-bins">Nombre de bins:</label>
        <input class="js-range-slider" id="plop-bins" data-min="1" data-
max="50" data-from="30" data-step="1" data-grid="true" data-grid-num="9.8"
data-grid-snap="false" data-prettify-separator="," data-prettify-
enabled="true" data-keyboard="true" data-data-type="number"/>
      </div>
    </form>
  </div>
  <div class="col-sm-8">
    <div id="distPlot" class="shiny-plot-output" style="width: 100% ;
height: 400px"></div>
  </div>
</div>
```

What do we test?

Once the UI is set, save the html from the module in the test folder:

```
htmltools::save_html(mod_premier_elementui("plop"), "ui.html")
```

Then, in the tests:

```
test_that("first module", {
  premier_el <- mod_premier_elementui("plop")
  golem::expect_html_equal(premier_el, "ui.html")
})
```

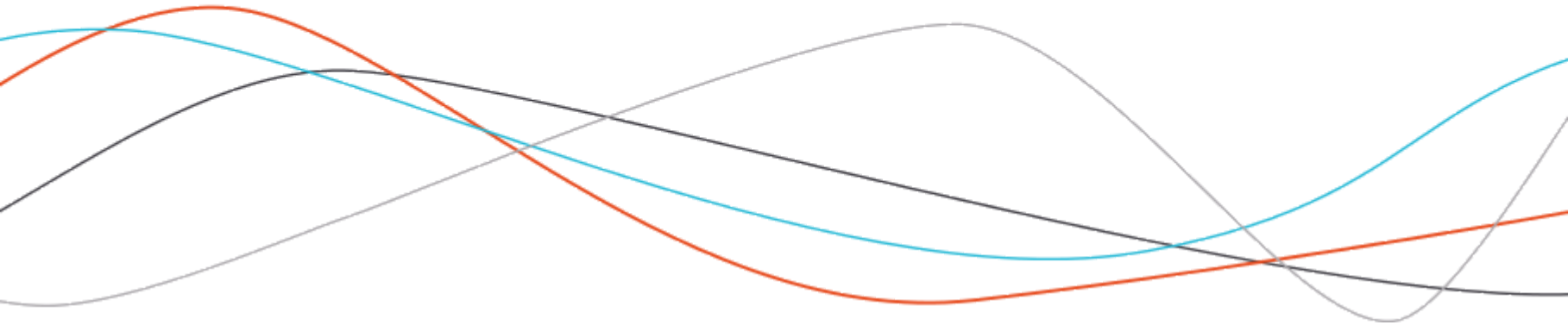
What do we test?

Does the app runs?

> Note that this will need specific config for GitLab and other CI tools.

```
context("launch")
library(processx)
testthat::test_that(
  "app launches",{
  # Launch the app as an external process
    x <- process$new(
      "R",  c( "-e", "setwd('../../'); pkgload::load_all();run_app()" )
    )
    # Let the app run
    Sys.sleep(5)
    # Check that the process is still alive
    expect_true(x$is_alive())
    x$kill()
  }
)
```

JS & CSS



Templates CSS & JS

```
golem::add_js_file("script")
golem::add_js_handler("script")
golem::add_css_file("custom")
```

> You don't need to specify the extensions

List them in `golem_add_external_resources()` from `app_ui()`.

Available in the app with, for example, `tags$img(src = "www/pics.jpeg")`

golem::js()

`golem` has a series of JavaScript functions that can be called from the server side.

These functions are there by default in the `app_ui`, with `golem::js()`.

They can be called with `session$sendCustomMessage("fonction", "reference_ui")`.

These functions all take `"reference_ui"`, referencing to the UI element you want to interact with. These can be either a jQuery selector for some functions, and for others the id or the class.

golem::js()

- `showid` & `hideid`, `showclass` & `hideclass` use the id or class ref to show and hide things.

```
session$sendCustomMessage("showid", ns("plot"))
```

- `showhref` & `hidehref`, same, but try to match an `href`

```
session$sendCustomMessage("showhref", "panel2")
```

- `clickon` clicks on an element. Needs to receive the full jQuery selector.
- `show` & `hide` hide or show an element. Needs to receive the full jQuery selector.

Some jQuery selectors

- `#plop`: element with id `plop`
- `.pouet`: element of class `pouet`
- `"button:contains('Afficher')"`: buttons that contain "Afficher".

HTML elements have attributes. For example:

```
<a href = "https://thinkr.fr" data-value = "panel2">ThinkR</a>
```

has `href` and `data-value`. We can refer to these attributes inside the jQuery selector, by putting `[]` after the tag name.

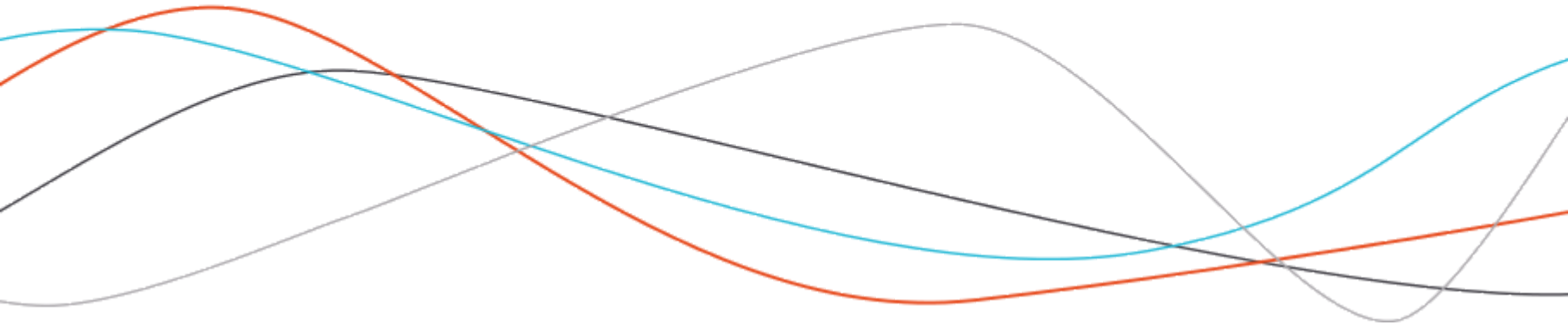
- `a[href = "https://thinkr.fr"]`: link with `href` == `https://thinkr.fr`
- `a[data-value="panel2"]`: link where `data-value` == `"panel2"`

Documentation

```
# Vignette
usethis::use_vignette("bing")
devtools::build_vignettes()
```

```
# Code coverage
usethis::use_travis()
usethis::use_appveyor()
usethis::use_coverage()
```

Deploy on Connect



Deploy

> After a first manual deployment

```
library(attempt)

secure_send_connect <- function( where = "." ) {
  res <- devtools::check(where)
  stop_if_not(res$errors, ~length(.x) == 0, "Erreur, app non déployée")
  rsconnect::deployApp(forceUpdate = TRUE)
}

secure_send_connect()
```

Thx! Questions?

Colin Fay

colin@thinkr.fr

http://twitter.com/_colinfay

http://twitter.com/thinkr_fr

https://github.com/ColinFay

https://thinkr.fr/

https://rtask.thinkr.fr/

https://colinfay.me/