Engineering {shiny} with {golem}

EARL 2020

2020-10-15

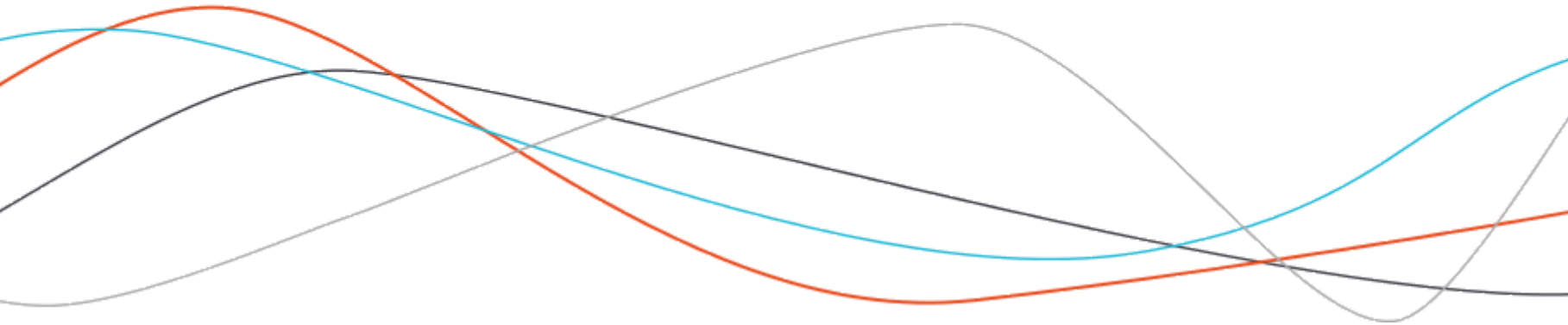
Colin Fay - ThinkR

Today's menu

Program

- Introduction
- Understanding `{golem}`
- Developing a `{golem}` app
- Testing & Sending to prod

INTRODUCTION



$ whoami

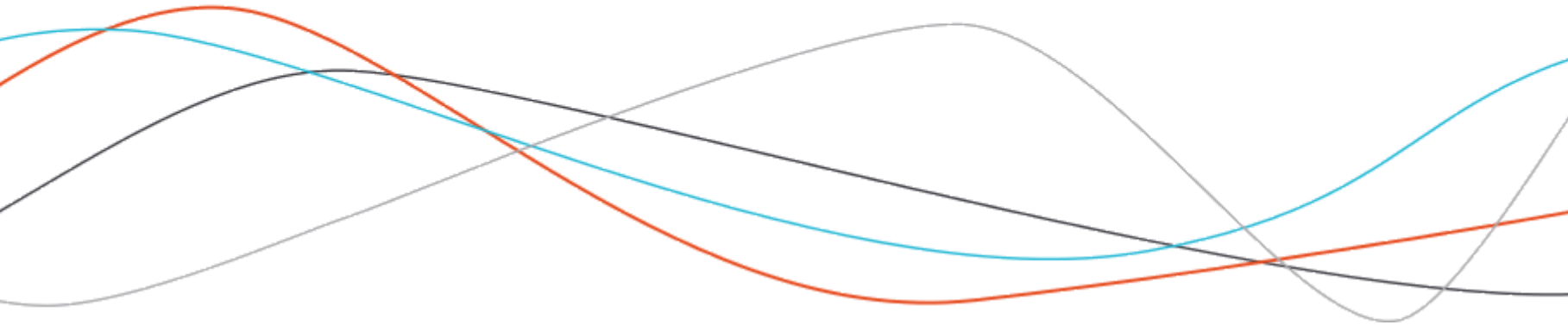
Colin FAY

Data Scientist & R-Hacker at ThinkR, a french company focused on Data Science & R.

Hyperactive open source developer, lead developer of the `{golem}` project.

- https://thinkr.fr
- https://rtask.thinkr.fr
- https://twitter.com/_colinfay
- https://github.com/colinfay
- https://colinfay.me

ThinkR



ThinkR

Data Science engineering, focused on R.

- Training
- Software Engineering
- R in production
- Consulting

Logistics

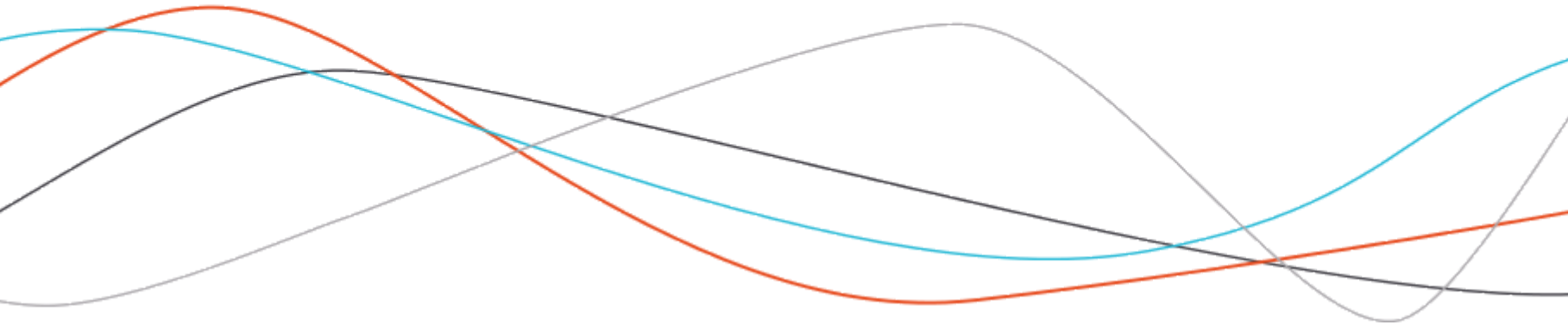
🔗 https://speakerdeck.com/colinfay

🔗 https://connect.thinkr.fr/make-a-golem

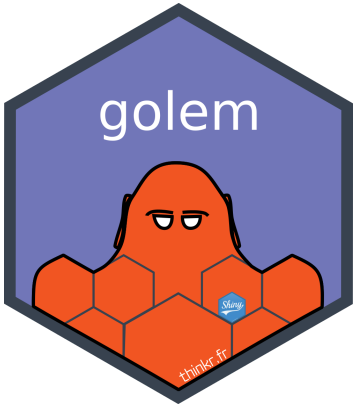
🐤 @_ColinFay

🐤 @earlconf

ABOUT GOLEM



{golem}



`{golem}` is an **R package** that contains a framework for building production-ready Shiny Applications.

```
golem <- cranlogs::cran_downloads(
  "golem",
  from = "2019-08-01"
)
sum(golem$count)
```

```
[1] 40295
```

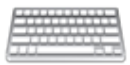
Why {golem}?

Why using `{golem}`?

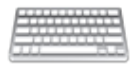
- Automate the ~~boring stuff~~ repetitive tasks
- Work with reliable tools
- Gain time developing
- Simplify deployment
- Standardize team work

About `{golem}` at ThinkR:

- Internal need, used on a daily basis
- Need reliable tooling for deploying to our clients' environments
- Build and share good practices globally
- Promote R & Shiny in production



Answer in the chat



What makes a production-grade `{shiny}` app?

Shiny App As a Package 

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

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

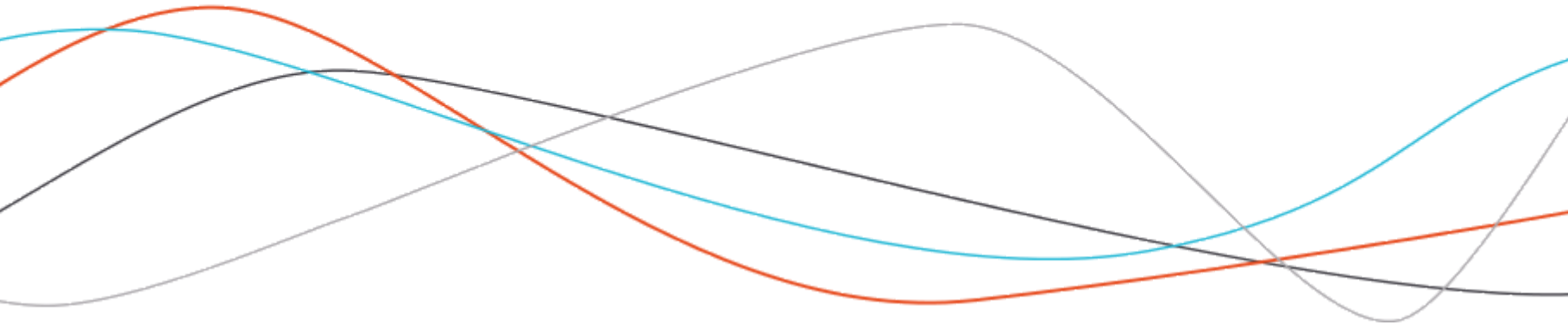
Shiny App As a Package 

The plus side: everything you know about package development works with `{golem}`.

Notably:

- Documentation
- Testing

Understanding {golem}



⚠ Warning ⚠

```
packageVersion("golem")
```

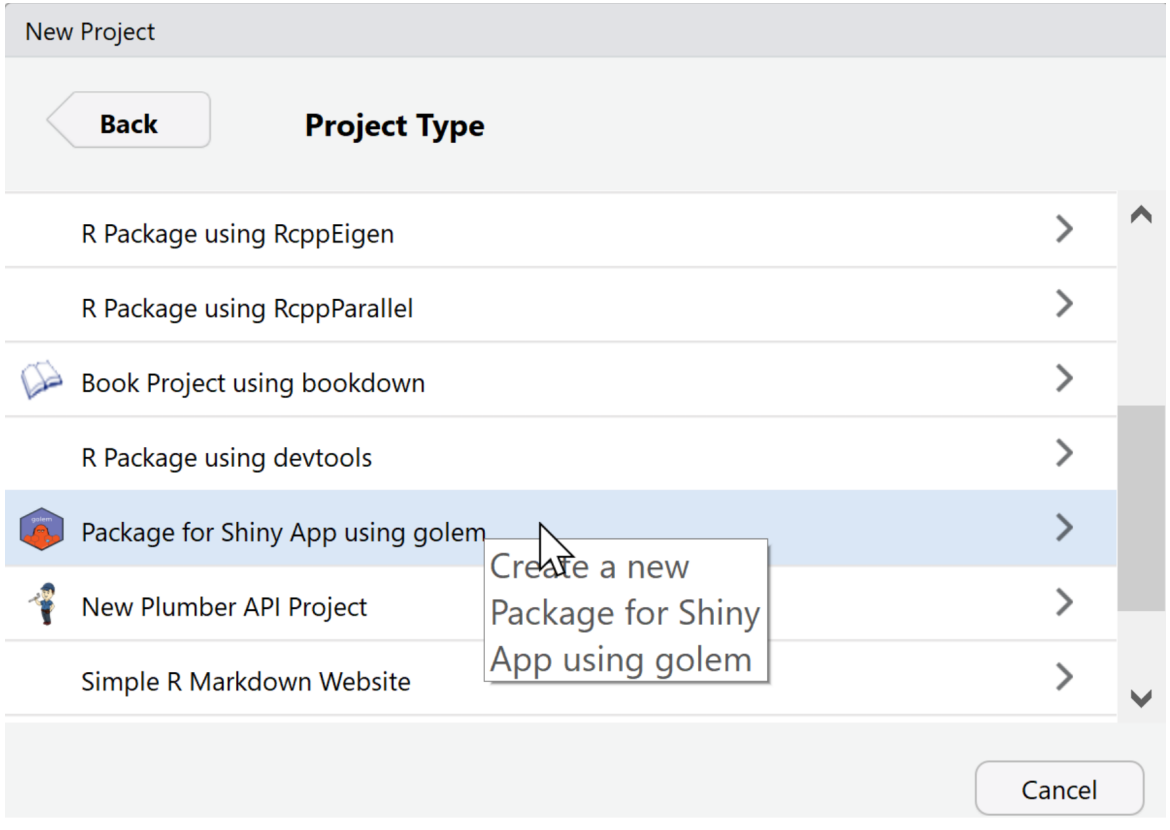
```
[1] '0.3.0'
```

```
remotes::install_github(
  "thinkr-open/golem"
)
```

> https://connect.thinkr.fr/make-a-golem / Ex 1

02:00

Create a {golem}



> https://connect.thinkr.fr/make-a-golem / Ex

02:00

Understanding {golem}

```
fs::dir_tree("golex")
```

```
golex
├── DESCRIPTION
├── NAMESPACE
├── R
|   ├── app_config.R
|   ├── app_server.R
|   ├── app_ui.R
|   └── run_app.R
├── dev
|   ├── 01_start.R
|   ├── 02_dev.R
|   ├── 03_deploy.R
|   └── run_dev.R
├── inst
|   ├── app
|   |   └── www
|   |       └── favicon.ico
|   └── golem-config.yml
└── man
    └── run_app.Rd
```

Understanding {golem}

Standard package files (i.e. not `{golem}`-specific):

- `DESCRIPTION`: meta-data
- `NAMESPACE`: exported functions + functions from other 📦
- `R/`: functions (everything in `{golem}` is a function)
- `man/`: documentation

app_server.R

```
#' The application server-side
#'
#' @param input,output,session Internal parameters for {shiny}.
#'     DO NOT REMOVE.
#' @import shiny
#' @noRd
app_server <- function( input, output, session ) {
  # Your application server logic

}
```

- server logic
- can be thought of as a drop in replacement of `server.R`
- series of `callModule()` / `module_server()` (if used)

app_ui.R

```
#' @param request Internal parameter for `{shiny}`.
#'     DO NOT REMOVE.
#' @import shiny
#' @noRd
app_ui <- function(request) {
  tagList(
    # Leave this function for adding external resources
    golem_add_external_resources(),
    # Your application UI logic
    fluidPage(
      h1("golex")
    )
  )
}
```

- UI counterpart
- put UI content after the `# Your application UI logic` line

app_ui.R

```
golem_add_external_resources <- function(){

  add_resource_path(
    'www', app_sys('app/www')
  )

  tags$head(
    favicon(),
    bundle_resources(
      path = app_sys('app/www'),
      app_title = 'golex'
    )
    # Add here other external resources
    # for example, you can add shinyalert::useShinyalert()
  )
}
```

- Used to add external resources
- Will integrate CSS and JS in the app

app_config.R

```
#' Access files in the current app
#'
#' NOTE: If you manually change your package name in the DESCRIPTION,
#' don't forget to change it here too, and in the config file.
#' For a safer name change mechanism, use the `golem::set_golem_name()` function.
#'
#' @param ... character vectors, specifying subdirectory and file(s)
#' within your package. The default, none, returns the root of the app.
#'
#' @noRd
app_sys <- function(...){
  system.file(..., package = "golex")
}
```

- `app_sys("x")` will refer to `inst/x`

app_config.R

```
get_golem_config <- function(
  value,
  config = Sys.getenv("R_CONFIG_ACTIVE", "default"),
  use_parent = TRUE
){
  config::get(
    value = value,
    config = config,
    # Modify this if your config file is somewhere else:
    file = app_sys("golem-config.yml"),
    use_parent = use_parent
  )
}
```

- Retrieves elements from `inst/golem-config.yml`

golem-config.yml

```
  golem_version: 0.0.0.9000
  app_prod: no
production:
  app_prod: yes
dev:
```

- Uses the `{config}` format
- Can be safely ignored if you don't feel like you need it 🤷‍♀️

run_app.R

```
#' @param ... arguments to pass to golem_opts
#' @inheritParams shiny::shinyApp
#'
#' @export
#' @importFrom shiny shinyApp
#' @importFrom golem with_golem_options
run_app <- function(
  onStart = NULL,
  options = list(),
  enableBookmarking = NULL,
  ...
) {
  with_golem_options(
    app = shinyApp(
      ui = app_ui,
      server = app_server,
      onStart = onStart,
      options = options,
      enableBookmarking = enableBookmarking
    ),
    golem_opts = list(...)
  )
```

> About `with_golem_options`

- Allows to pass arguments to `run_app()`
- will later be callable with `golem::get_golem_options()`

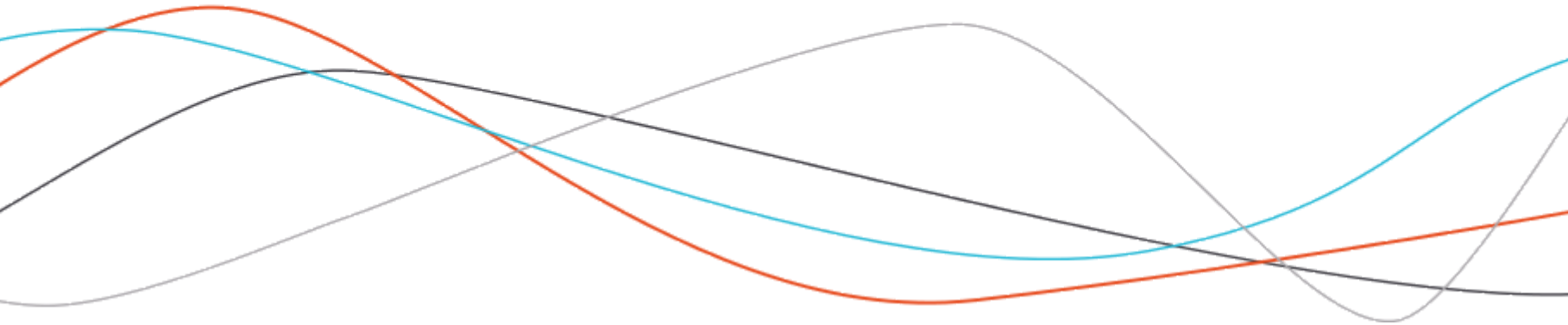
Understanding {golem}

`inst/app/www/`

Host external files, notably the one created with:

- `golem::add_css_file()`
- `golem::add_js_file()`
- `golem::add_js_handler()`
- `golem::add_js_input_binding()`
- `golem::add_js_output_binding()`
- `golem::use_favicon()`
- `golem::add_html_template()`

About the dev/ folder



About the dev/ folder

```
fs::dir_tree("golex/dev")
```

```
golex/dev
├── 01_start.R
├── 02_dev.R
├── 03_deploy.R
└── run_dev.R
```

Four files that bundle the golem workflow:

- `01_start.R`: run once at the beginning of the project
- `02_dev.R`: day to day development
- `03_deploy.R`: to use before sending to prod
- `run_dev.R`: to relaunch your app during development

`01_start.R`

- Fill the `DESCRIPTION` file

> https://connect.thinkr.fr/make-a-golem / Ex 3

05:00

`01_start.R`

To be launched for setting elements:

- `golem::set_golem_options()`
  Fills the `yaml` file
- `golem::use_recommended_tests()` and `golem::use_recommended_deps()`
  Sets common dependencies and tests
- Use `golem::set_golem_name()`
  to globally change your app name

01_start.R

{usethis} commonly used calls

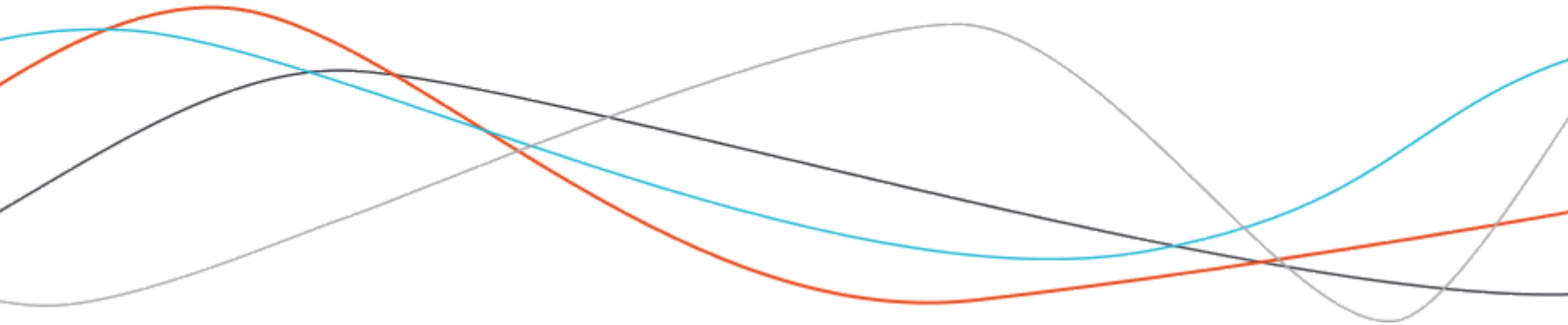
```
usethis::use_mit_license( name = "Golem User" )  # You can set another
license here
usethis::use_readme_rmd( open = FALSE )
usethis::use_code_of_conduct()
usethis::use_lifecycle_badge( "Experimental" )

usethis::use_news_md( open = FALSE )
usethis::use_git()
```

`01_start.R`

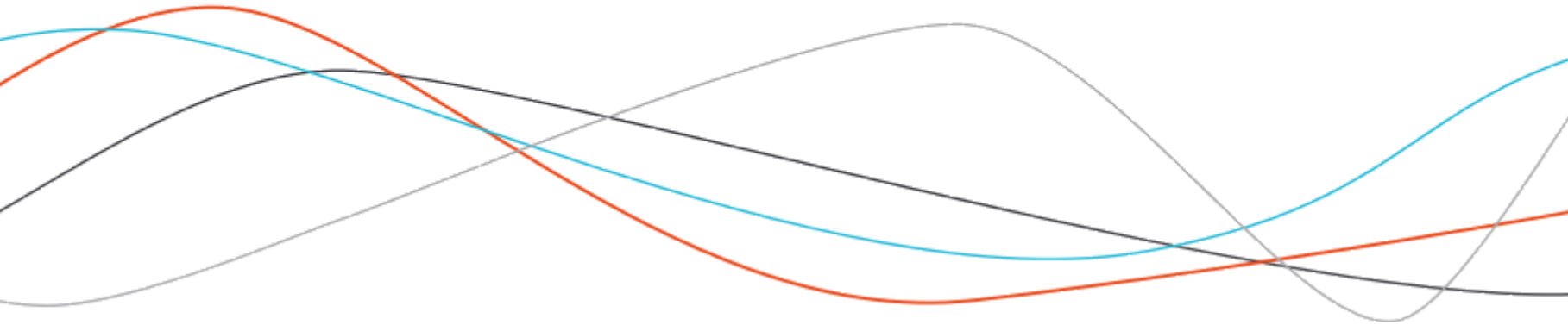
- `golem::use_recommended_deps()`
- `golem::use_favicon()`
- `golem::use_utils_ui()` & `golem::use_utils_server()`

> https://connect.thinkr.fr/make-a-golem / Ex 4

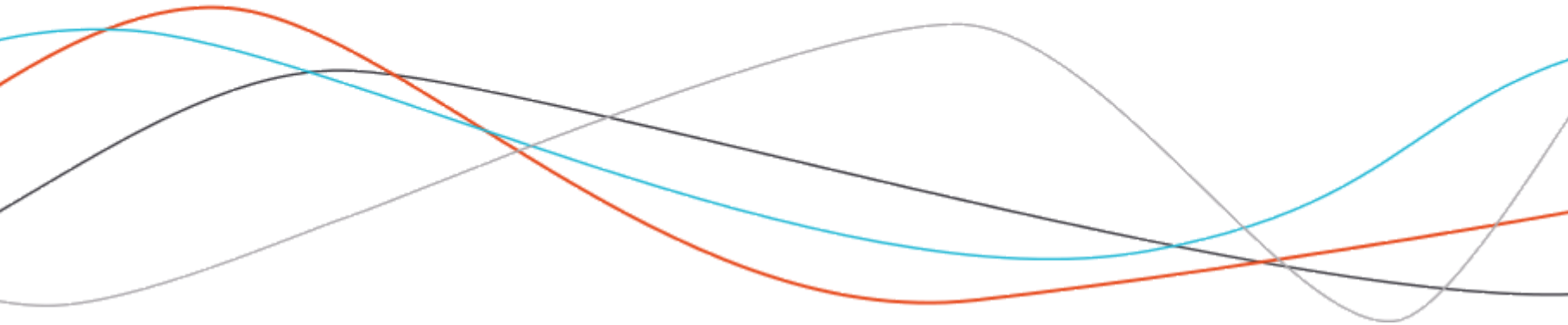
DEV



STRUCTURING YOUR APPLICATION

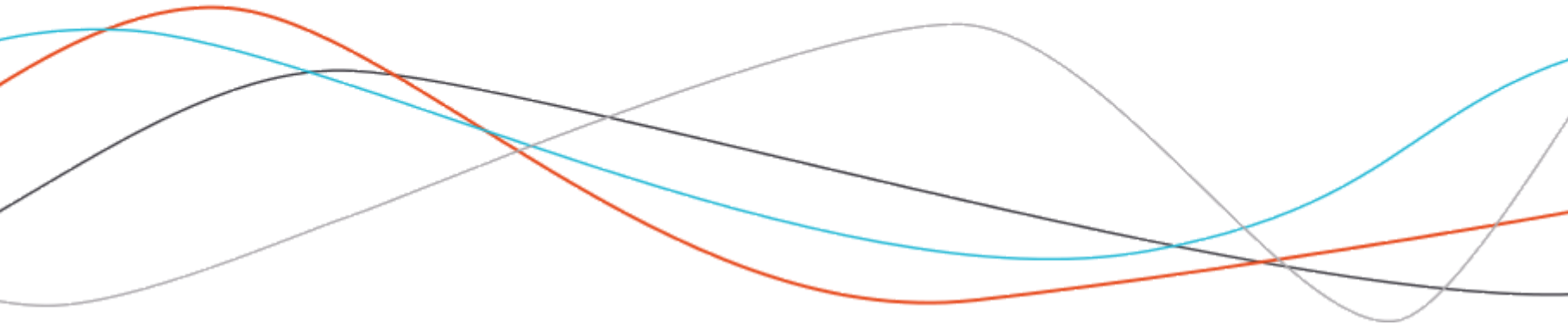


ABOUT DATA



⌨ Answer in the chat ⌨

What is the usual source of your data when building an app?



Adding datasets

- You might need internal data inside your app
- Call `usethis::use_data_raw` from `dev/02_dev.R`

```
## Add internal datasets ----
## If you have data in your package
usethis::use_data_raw( name = "dataset" )
```

- Creates `data-raw/dataset.R`
- Inside this file:

```
## code to prepare `dataset` dataset goes here

usethis::use_data("dataset")
```

Adding datasets

```
## code to prepare `dataset` dataset goes here
library(tidyverse)
my_app_dataset <- read.csv("bla/bla/bla.csv")
my_app_dataset <- my_app_dataset %>%
  filter(this == "that") %>%
  arrange(on_that) %>%
  select( -contains("this") )
usethis::use_data(my_app_dataset)
```

Now available as `my_app_dataset` inside your app.

External data (database)

- If your app is large or changes frequently, it's better to use a database as a backend.

Choice	Update	Size
Package data	Never to very rare	Low to medium
Reading files	Uploaded by Users	Preferably low
External DataBase	Never to Streaming	Low to Big

Adding data in your app

> https://connect.thinkr.fr/make-a-golem / Ex 5

05:00

Application logic vs business logic

- Application logic: the things that make your Shiny app interactive
- Business logic: core algorithms and functions that make your application specific to your area of work

Keep them separated!

Structuring your app

Naming convention:

- `app_*`: global infrastructure functions
- `fct_*`: business logic functions
- `mod_*`: file with ONE module (ui + server)
- `utils_*`: cross module utilitarian functions
- `*_ui_*` / `*_server_*`: relates to UI and SERVER

Why bother?

- 🤔: `R/connect.R`
- 🤔: `R/summary.R`
- 🤔: `R/plot.R`
- 🤔: `R/odbc.R`

Why bother?

- 🤔: `R/connect.R`
- 🤔: `R/summary.R`
- 🤔: `R/plot.R`
- 🤔: `R/odbc.R`

- 🤩: `R/fct_connect.R`
- 🤩: `R/mod_summary.R`
- 🤩: `R/utils_plot.R`
- 🤩: `R/fct_odbc.R`

02_dev.R

```
golem::add_fct( "helpers" )
golem::add_utils( "data_ui" )
```

```
✓ File created at R/fct_helpers.R
• Go to R/fct_helpers.R

✓ File created at R/utils_data_ui.R
• Go to R/utils_data_ui.R
```

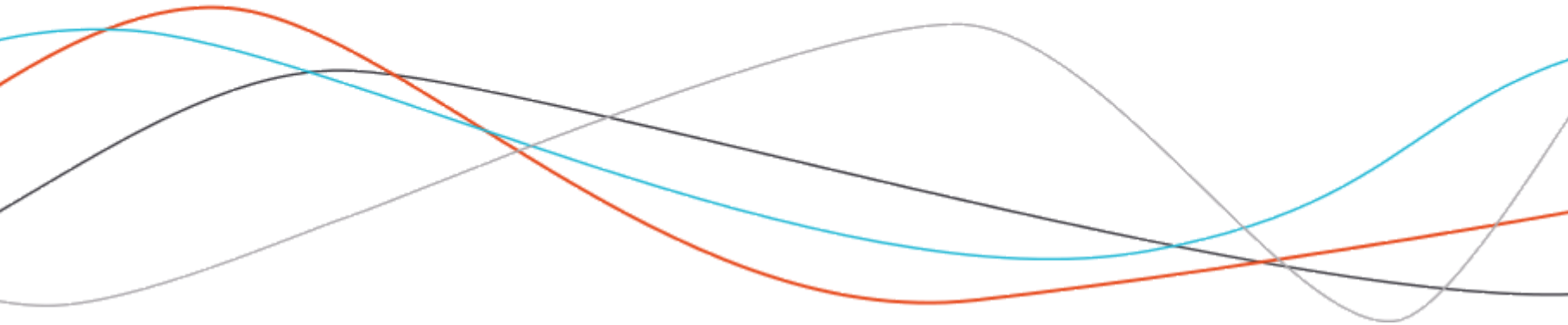
Creates `fct_*` and `utils_*` files

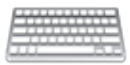
```
golem::add_module( name = "my_first_module")
```

```
✓ File created at R/mod_my_first_module.R
• Go to R/mod_my_first_module.R
```

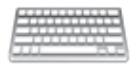
Builds a skeleton for a Shiny Module

USING SHINY MODULES

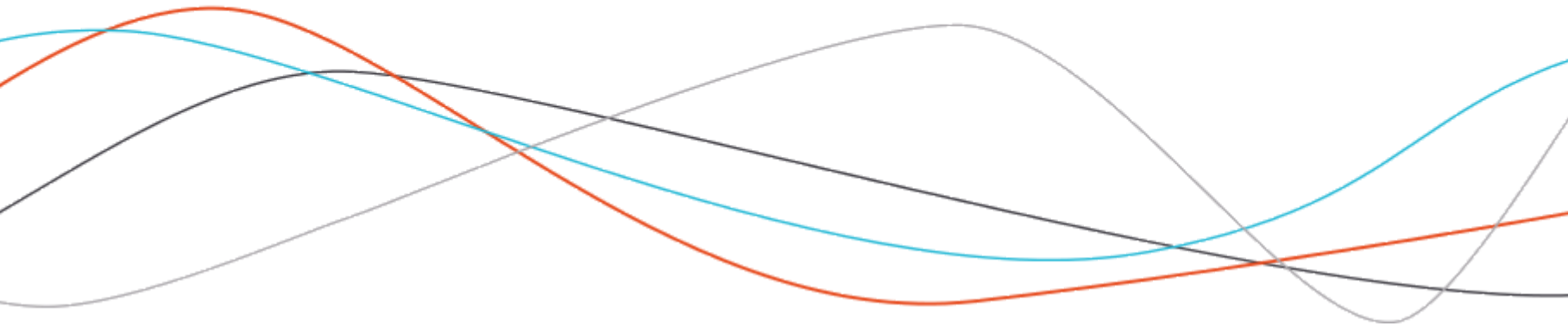




Answer in the chat



Have you ever used `{shiny}` modules?



About Shiny Modules

- "Pieces" of Shiny Apps
- Always come in pair (UI + Server)
- Manage the unique IDs necessity of Shiny
- "Functionnalize" your app

One million "Validate" buttons

```
library(shiny)
ui <- function(request){
  fluidPage(
    sliderInput("choice1", "choice 1", 1, 10, 5),
    actionButton("validate1", "Validate choice 1"),
    sliderInput("choice2", "choice 2", 1, 10, 5),
    actionButton("validate2", "Validate choice 2")
  )
}
server <- function(input, output, session){
  observeEvent( input$validate1 , {
    print(input$choice1)
  })
  observeEvent( input$validate2 , {
    print(input$choice2)
  })
}
shinyApp(ui, server)
```

Why Shiny Modules

Functionalizing Shiny elements:

```
name_ui <- function(id){
  ns <- NS(id)
  tagList(
    sliderInput(ns("choice"), "Choice", 1, 10, 5),
    actionButton(ns("validate"), "Validate Choice")
  )
}

name_server <- function(id){
  moduleServer(id, function(input, output, session) {
    observeEvent( input$validate , {
      print(input$choice)
    })
  })
}
```

Why Shiny Modules

Functionalizing Shiny elements:

```
library(shiny)
ui <- function(request){
  fluidPage(
    name_ui("name_ui_1"),
    name_ui("name_ui_2")
  )
}

server <- function(input, output, session){
  name_server("name_ui_1")
  name_server("name_ui_2")
}

shinyApp(ui, server)
```

Why Shiny Modules

```
id <- "name_ui_1"
ns <- NS(id)
ns("choice")
```

```
[1] "name_ui_1-choice"
```

```
name_ui <- function(id, butname){
  ns <- NS(id)
  tagList( actionButton(ns("validate"), butname) )
}

name_ui("name_ui_1", "Validate Choice")
name_ui("name_ui_2", "Validate Choice, again")
```

```
<button id="name_ui_1-validate" type="button" class="btn btn-default action-
button">Validate Choice</button>
```

```
<button id="name_ui_2-validate" type="button" class="btn btn-default action-
button">Validate Choice, again</button>
```

02_dev.R

```
golem::add_module( name = "my_first_module")
```

Builds a skeleton for a Shiny Module

02_dev.R

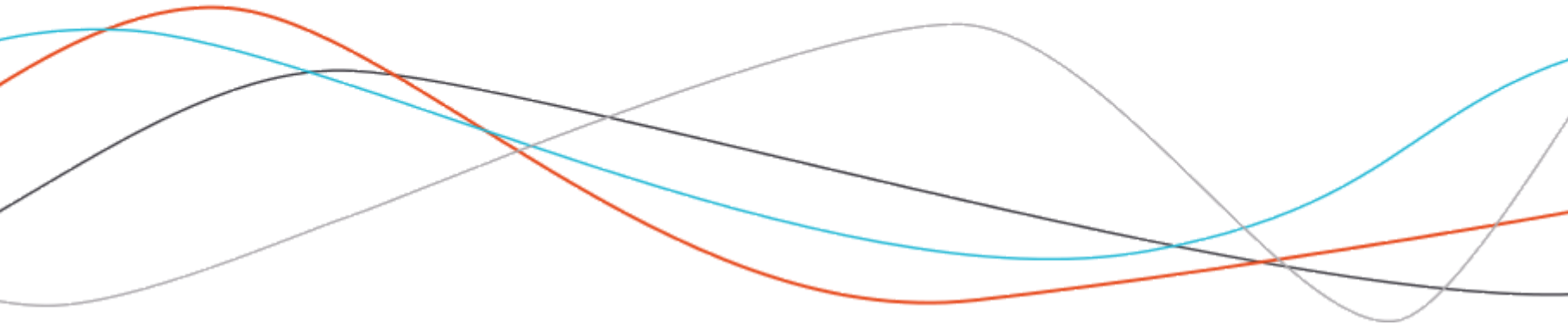
```
#' my_first_module UI Function
#'
#' @description A shiny Module.
#'
#' @param id,input,output,session Internal parameters for {shiny}.
#'
#' @noRd
#'
#' @importFrom shiny NS tagList
mod_my_first_module_ui <- function(id){
  ns <- NS(id)
  tagList(

  )
}
```

02_dev.R

```
#' my_first_module Server Functions
#'
#' @noRd
mod_my_first_module_server <- function(id){
  moduleServer( id, function(input, output, session){
    ns <- session$ns

  })
}

## To be copied in the UI
# mod_my_first_module_ui("my_first_module_ui_1")

## To be copied in the server
# mod_my_first_module_server("my_first_module_ui_1")
```

⚠️ DON'T FORGET THE NS() IN THE UI ⚠️

UNDERSTANDING NAMESPACE & DEPENDENCIES



About the NAMESPACE file

- One of the most important files of your package
- ⚠ NEVER EDIT BY HAND ⚠
- Describes **how your package interacts with R, and with other packages**
- Lists functions that are exported (from your app) and imported (from other packages)

What's a dependency?

- Your app needs external functions (at least from `{shiny}`)
- The `DESCRIPTION` file contains the package dependencies
- Are added to the `DESCRIPTION` with:

```
usethis::use_package("attempt")
```

What's a dependency?

- You also need to add tags on top of each functions that specify what deps are imported
- Either with `@import` (a whole package) and `@importFrom` (a specific function).

> golem built modules, by default, import elements from `{shiny}`:

```
#' @importFrom shiny NS tagList
```

Do this for EACH function

```
#' @import magrittr
#' @importFrom  stats na.omit

mean_no_na <- function(x){
  x <- x %>% na.omit()
  sum(x)/length(x)
}
```

- You can use `import` or `importFrom`.
- The better is to use `importFrom`, for preventing namespace conflict.
- Add to EACH function.

> It will take some time, but it's better on the long run.

TO SUM UP

"I want to use dplyr::filter in my_module.R"

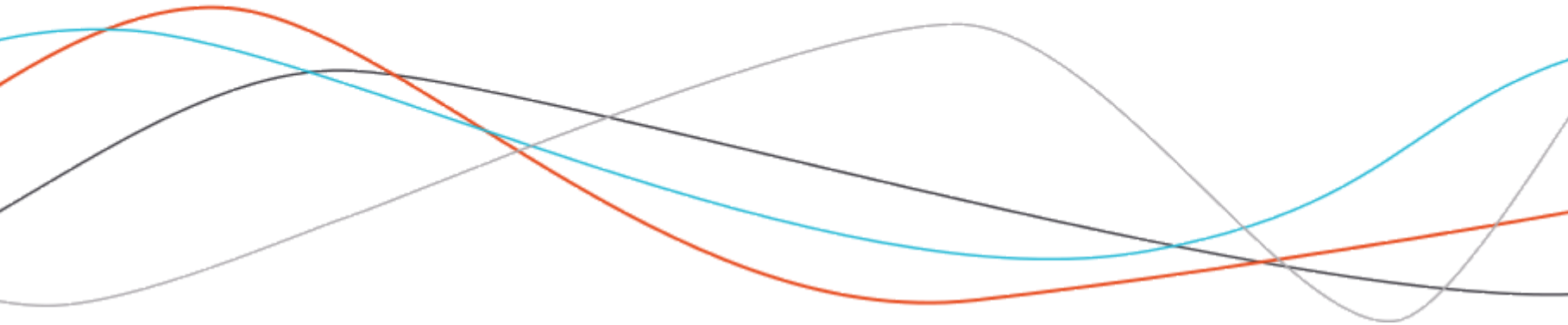
1. `usethis::use_package('dplyr')` in the console
2. `#' @importFrom dplyr filter` in the file
3. `devtools::document()`

Building your first module

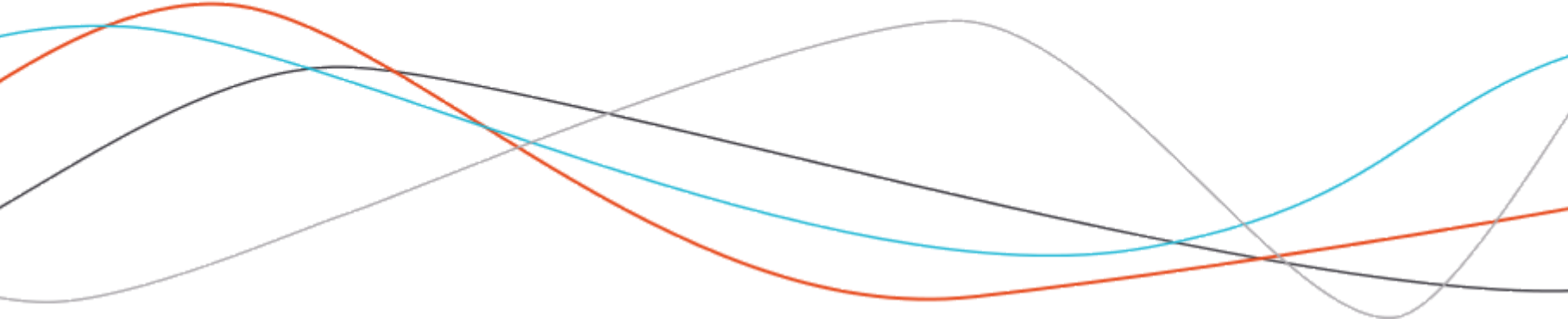
> https://connect.thinkr.fr/make-a-golem / Ex 6

15:00

INCLUDING EXTERNAL FILES



```
golem_add_external_resources()
```



golem_add_external_resources()

- In `app_ui.R`, the `golem_add_external_resources()` functions add to the app every `.css` and `.js` file contained in `inst/app/www`

```
golem_add_external_resources <- function(){

  add_resource_path(
    'www', app_sys('app/www')
  )

  tags$head(
    favicon(),
    bundle_resources(
      path = app_sys('app/www'),
      app_title = 'golex'
    )
    # Add here other external resources
    # for example, you can add shinyalert::useShinyalert()
  )
}
```

Use external resources

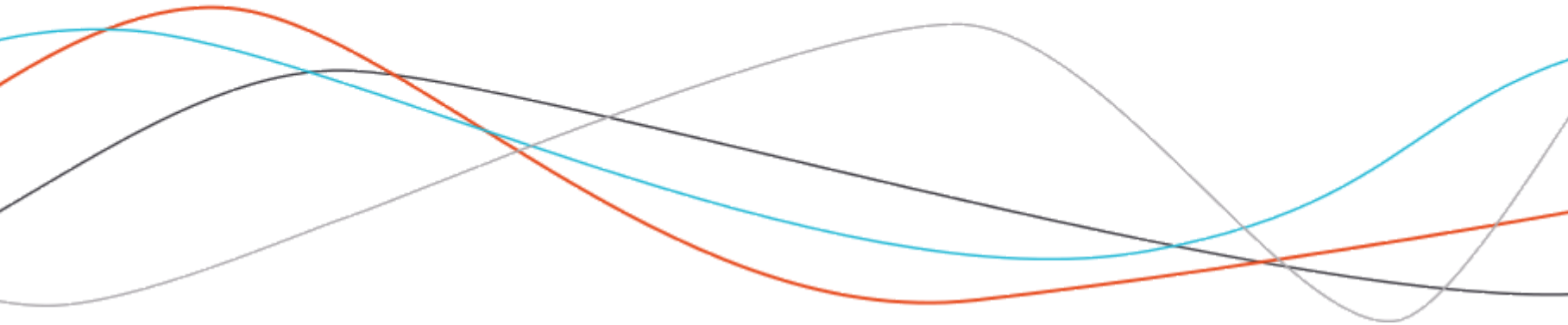
- `golem::add_css_file()`
- `golem::add_js_file()`
- `golem::add_js_handler()`
- `golem::add_js_input_binding()`
- `golem::add_js_output_binding()`
- `golem::use_favicon()`
- `golem::add_html_template()`

Add CSS to your app

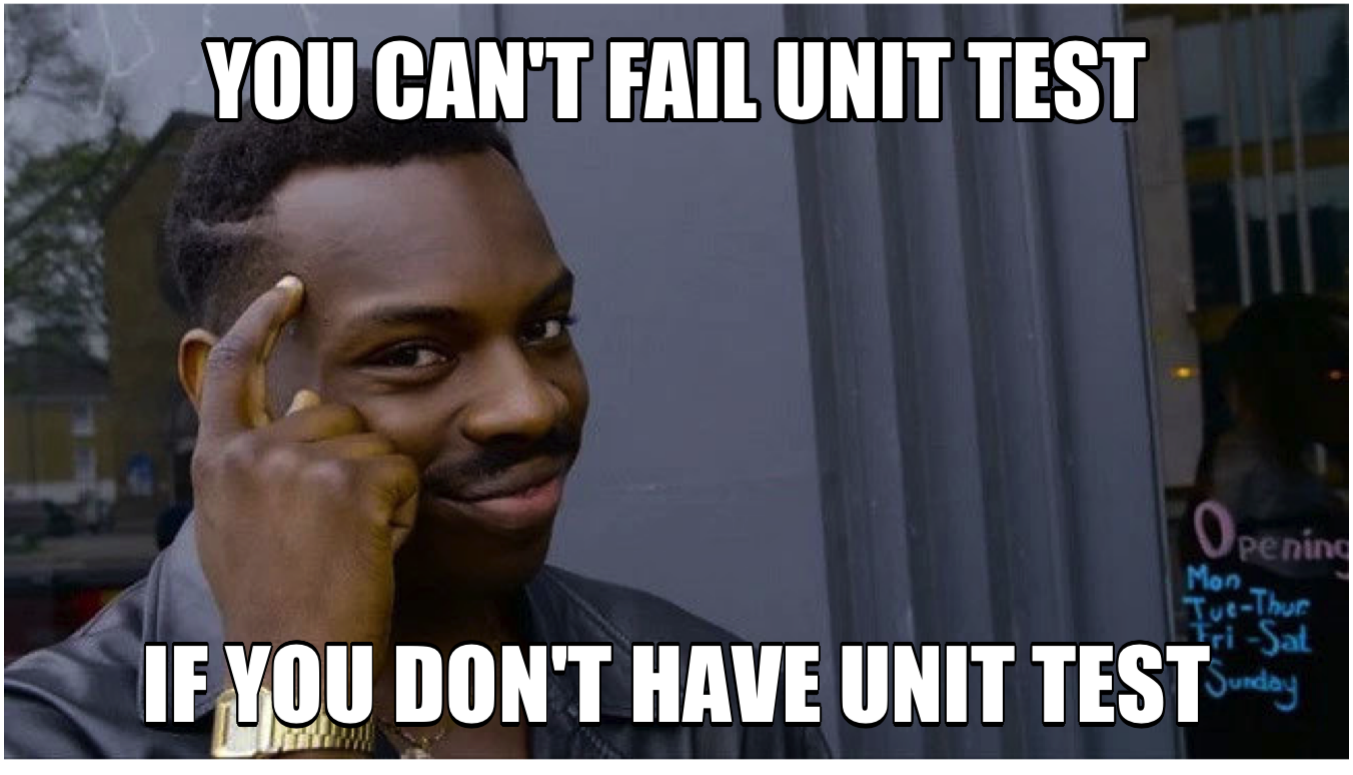
> https://connect.thinkr.fr/make-a-golem / Ex 7

05:00

TEST AND SEND



Using unit tests



What do we test

- Focus on business logic testing: it's more important to test that the algorithm is still accurate than testing the UI
- As we've separated businness logic from application logic, we use package development testing tools
- We'll use the `{testthat}` 📦

01_start.R

```
## Init Testing Infrastructure ----
## Create a template for tests
golem::use_recommended_tests()
```

> Recommended tests

02_dev.R

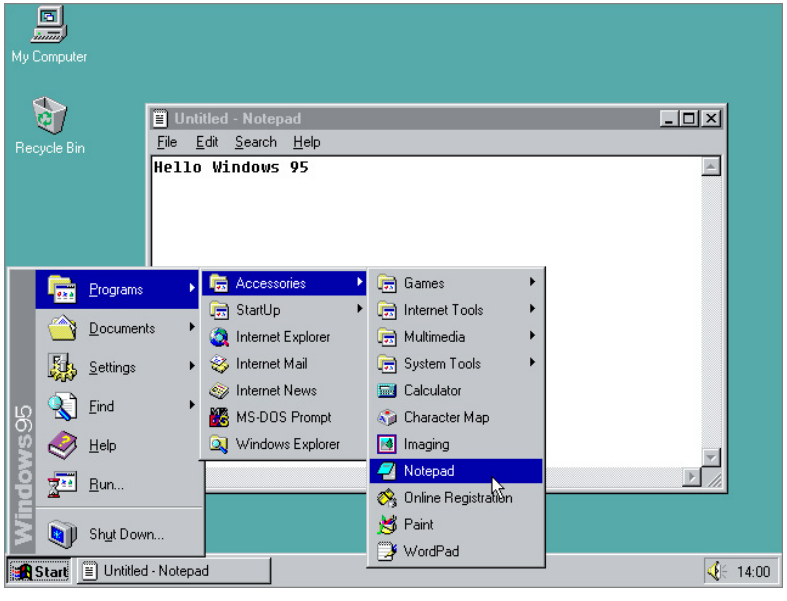
```
## Tests ----
## Add one line by test you want to create
usethis::use_test( "app" )
```

> Add custom test

What - User Interface

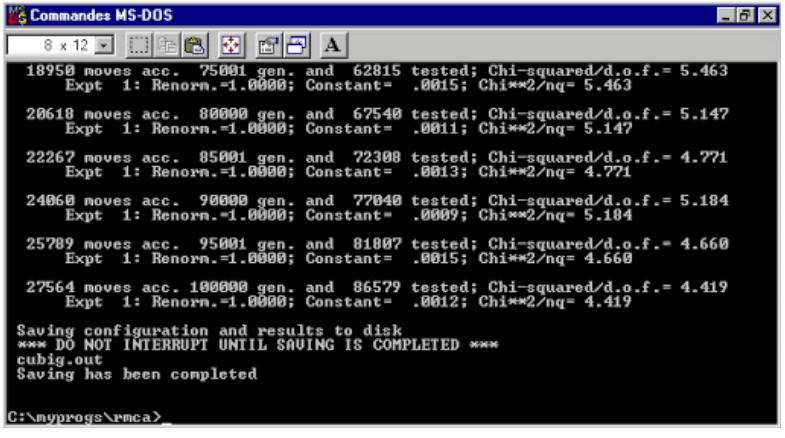
> Your time is limited, so if you have to choose don't focus too much on testint the UI only

- What the users see
- What the users interact with
- General front-end/design



What - Business logic

> Try to test business logic as extensively as possible



- Core algorithms that make your app "unique"
- Business knowledge
- What your users rely on

What - Application Load

> Poor app performances lead to bad UX, and potentially cost 💸

- How much CPU & RAM does your application need
- Bad estimate will lead to slow application performances
- If the app needs to scale, it's crucial to know it upfront

Shiny App as a package 📦

-> Leverage standard testing frameworks

```
test_that("The meaning of life is
42", {
  expect_equal(
    meaning_of_life(),
    42
  )
})
```

UI regressions

`{shinytests}`

- 👓 Test visual regression of your application

`puppeteer`

- 🎡 Command line tool to mock a web session, in NodeJS

`{crrry}`

- 🚗 R tool to drive a `{shiny}` session

Testing the app load

`{shinyloadtest}`

- 🏋️: native R package + Cli to record and replay load tests

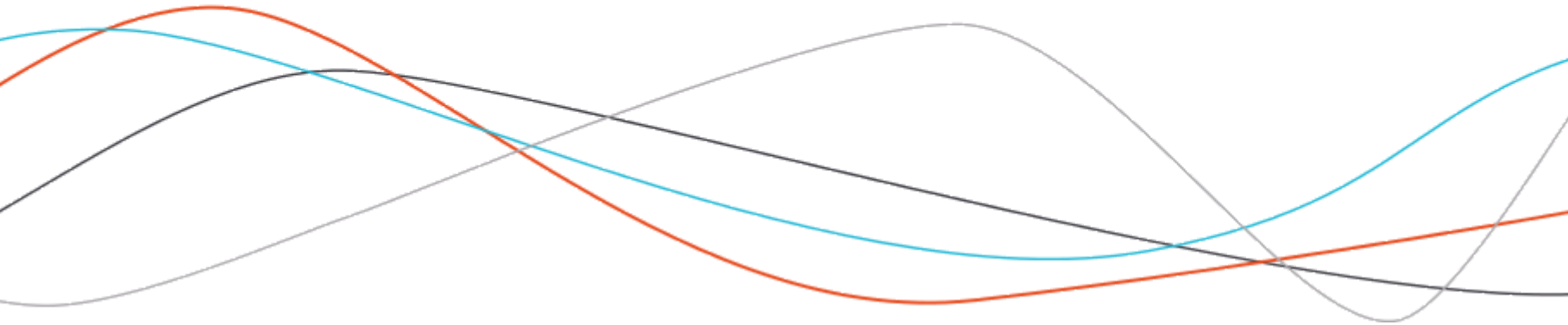
`{dockerstats}`

- 🐋: get Docker stats inside R

`{crrry}` + `{dockerstats}`

- 👨‍👧: replay session and watch the Docker stats

SEND TO PRODUCTION



Send to prod

- Once everything is tested, you can send to prod using `{golem}`
- Deploy on a server, or build as a `tar.gz` with `pkgbuild::build()`

```
## RStudio ----
## If you want to deploy on RStudio related platforms
golem::add_rstudioconnect_file()
golem::add_shinyappsio_file()
golem::add_shinyserver_file()
```

```
## Docker ----
## If you want to deploy via a generic Dockerfile
golem::add_dockerfile()
## If you want to deploy to ShinyProxy
golem::add_dockerfile_shinyproxy()
## If you want to deploy to Heroku
golem::add_dockerfile_heroku()
```

Deploy

> https://connect.thinkr.fr/make-a-golem / Ex 8

05:00

engineering-shiny.org

Engineering Production-Grade Shiny Apps

Colin Fay, Sébastien Rochette, Vincent Guyader, Cervan Girard

2020-04-25

Introduction

This book is currently under development. It will be published in 2020 in the R Series by Chapman & Hall.

Motivation

This book will not **get you started with Shiny**, nor **talk about how to deploy into production and scale your app**. What we'll see is **the process of building the app**. Why? Lots of blog posts and books talk about starting to use `{shiny}` (**???**) or putting apps in production. Very few (if any) talk about this grey area between getting started and pushing into production.

So this is what this book is going to talk about: building Shiny application. We'll focus on the process, the workflow, and the tools we use at ThinkR when building big Shiny Apps.

Hence, if you are starting to read this book, we assume you have a working knowledge of how to build a small application, and want to know how to go one step further.

Thx! Questions?

Colin Fay

Online

- 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/