Introduction to {golem}

Shiny Korea meet-up

2022-06-15

조호연

Building tidy tools

Should I take this course?

You should take this workshop if you have experience programming in R and want to learn how to tackle larger scale problems. You'll get the most from it if you're already familiar with functions and are comfortable with R's basic data structures (vectors, matrices, arrays, lists, and data frames). Note: There is ~30% overlap in the material with Hadley's previous "R Masterclass". However, the material has been substantially reorganized, so if you've taken the R Masterclass in the past, you'll still learn a lot in this class.

What will I learn?

This course has three primary goals. You will:

- Learn efficient workflows for developing high-quality R functions, using the set of conventions codified by a package. You'll also learn workflows for unit testing, which helps ensure that your functions do exactly what you think they do.

https://adv-r.hadley.nz/
https://r-pkgs.org/

Learning Shiny for Production – Remote session

Learning Shiny for Production

Table of C
1. Learning S
2. Shiny for P
2.1. Progra
2.2. Exam
2.3. Who is
2.4. Fees
2.5. Logist
2.6. Goodie
2.7. Regist
3. Shiny for P

Hello world!

We're very happy to announce that we will be **giving a remote training session on building Shiny Application for production in July**. Be quick, we only have 10 spots available!

If you always wanted to know how to **build Shiny applications by the rulebook**, if you already know R and want to create nice shaped and **maintenable shiny application**, this remote training session is for you (details and fees bellow).

Registrer Now

Colin Fay | Diane Beldame | Cervan Girard | Vincent Guyader | Sébastien Rochette | Margot Brard

https://engineering-shiny.org/

Agenda

- What & why {golem}
- Quick start guide
- App architecture
- Use cases & demo
- Miscellaneous

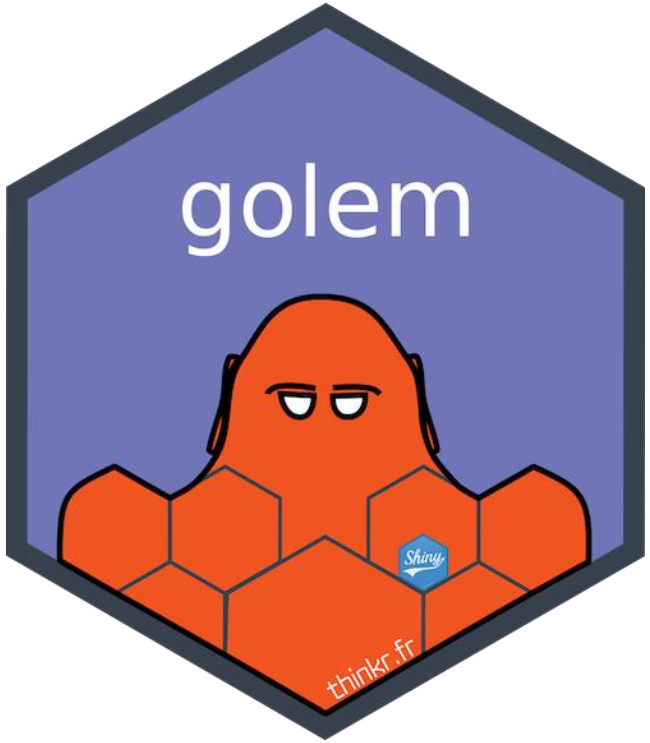
What & why {golem}

Before I found {golem}

Is there a better way?

- 튼튼한 모듈 설계를 쉽게 할 수 없을까
- 테스트를 조금 더 체계적으로 할 수 없을까
- 무거운 server-side processing을 덜 쓸 수는 없을까

Then I met {golem}



- 튼튼한 **<u>패키지 형태의 앱</u>**을 만드는 데에 유용한 툴킷
- 패키지 개발에 통용되는 {usethis}, {devtools} 의 기반 위에 만들어짐, 때문에 아래와 같은 패키지 개발의 장점 활용
  - Dependency 관리
  - Documentation
  - Testing 자동화
  - Meta data
- Javascript나 CSS를 쉽게 쓸 수 있게끔 함

단, 앱 유지/보수을 위해 패키지 개발에 대한 지식이 필요하다는 것은 downside

Quick start guide

How to start

Installation

- You can install the stable version from CRAN with:

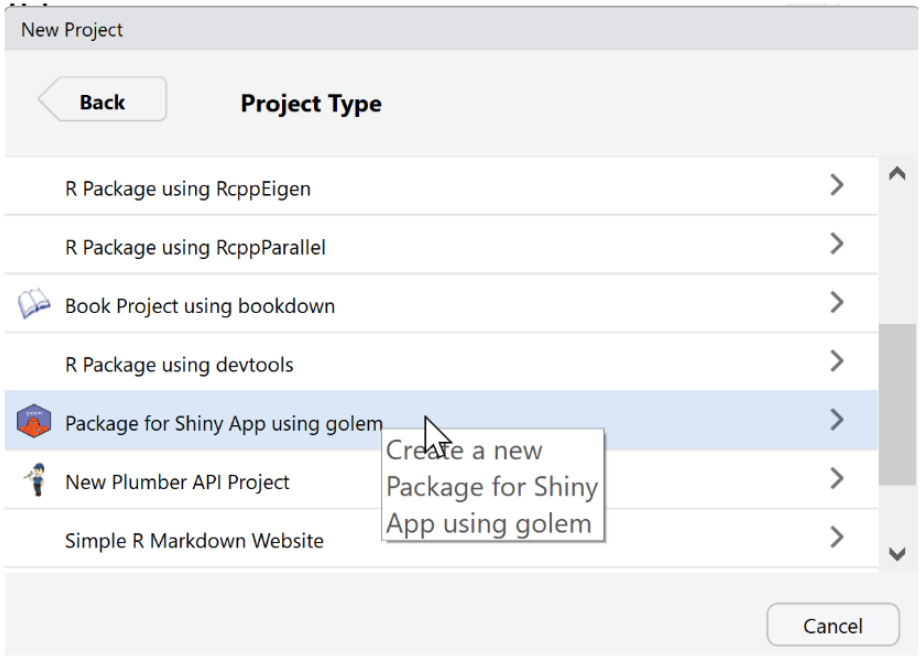
```
install.packages("golem")
```

- You can install the development version from GitHub with:

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

Launch the project

Create a new package with the project template:



Source: GitHub - ThinkR-open/golem: A Framework for Building Robust Shiny Apps

{golem} app structure

```
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
```

패키지 핵심 구성 요소

Production 버전 스크립트*

개발 가이드

앱에 필요한 function, module 등을 여기에서 생성

Dev 모드에서 앱 실행

패키지 시스템 폴더

Javascript, CSS 등 부차적 요소는 여기에 저장

패키지 문서

가장 빈번한 workflow:

02_dev.R 에서 필요한 구성 요소를 R 폴더에 생성한 후,

R 폴더의 해당 구성 요소를 개발해 나가는 식

그리고 run_dev.R 에서 테스트 모드 실행

3. Day-to-day dev with golem

A. Look at your golem

- **Launch your app with *dev/run_dev.R*:**

```
options(golem.app.prod = FALSE)
    Sets the prod or dev mode. (see ?golem::app_dev )
golem::detach_all_attached()
    Detaches all loaded packages and cleans your environment.
golem::document_and_reload()
    Documents and reloads your package.
appdemo::run_app()
    Launches your application.
```

B. Customise your golem with dev/02_dev.R

- **Edit R/app_ui.R & R/app_server.R**

  'R/app_ui.R' & 'R/app_server.R' hold the UI and server logic of your app. You can edit them directly, or add elements created with golem (e.g, modules).

- **Add shiny modules**

```
golem::add_module( name = "example" )
    Creates 'R/mod_example.R', with mod_example_ui and
    mod_example_server functions inside.
```

Source: https://engineering-shiny.org/golem.html#understanding-golem-app-structure

* 주의: R 폴더에는 완성된 스크립트만 있어야 함, 에러가 있을 경우 패키지를 로드할 수가 없음

Quick walkthrough

- Step-by-step demo
  https://youtu.be/3-p9XLvoJV0?t=90

- 또는 아래 vignette 참조
  - https://thinkr-open.github.io/golem/articles/a_start.html
  - https://thinkr-open.github.io/golem/articles/b_dev.html
  - https://thinkr-open.github.io/golem/articles/c_deploy.html

Architecture

Business logic과 app logic의 분리

분리가 안된 예

```
library(shiny)
library(dplyr)
# A simple app that returns a table
ui <- function() {
  tagList(
    tableOutput("tbl"),
    sliderInput("n", "Number of rows", 1, 50, 25)
  )
}

server <- function(input, output, session) {
  output$tbl <- renderTable({
    # Writing all the business logic for the table manipulation
    # inside the server
    mtcars %>%
      # [...] %>%
      # [...] %>%
      # [...] %>%
      # [...] %>%
      # [...] %>%
      top_n(input$n)
  })
}

shinyApp(ui, server)
```

분리가 된 예

```
library(shiny)
library(dplyr)

# Writing all the business logic for the table manipulation
# inside an external function
top_this <- function(tbl, n) {
  tbl %>%
    # [...] %>%
    # [...] %>%
    # [...] %>%
    # [...] %>%
    top_n(n)
}

# A simple app that returns a table
ui <- function() {
  tagList(
    tableOutput("tbl"),
    sliderInput("n", "Number of rows", 1, 50, 25)
  )
}


server <- function(input, output, session) {
  output$tbl <- renderTable({
    # We call the previously declared function inside the server
    # The business logic is thus defined outside the application
    top_this(mtcars, input$n)
  })
}

shinyApp(ui, server)
```

Source: https://engineering-shiny.org/structuring-project.html#business-logic-and-application-logic

Naming convention

- `app_*.R` (typically `app_ui.R` and `app_server.R`) contain the top-level functions defining your user interface and your server function.
- `fct_*` files contain the business logic, which are potentially large functions. They are the backbone of the application and may not be specific to a given module. They can be added using golem with the `add_fct("name")` function.
- `mod_*` files contain a unique module. Many shiny apps contain a series of tabs, or at least a tab-like pattern, so we suggest that you number them according to their step in the application. Tabs are almost always named in the user interface, so that you can use this tab name as the file name. For example, if you build a dashboard where the first tab is called "Import", you should name your file `mod_01_import.R`. You can create this file with a module skeleton using `golem::add_module("01_import")`.
- `utils_*` are files that contain utilities, which are small helper functions. For example, you might want to have a `not_na`, which is `not_na <- Negate(is.na)`, a `not_null`, or small tools that you will be using application-wide. Note that you can also create `utils` for a specific module.

Source: https://engineering-shiny.org/structuring-project.html#conventions-matter

모듈 간 커뮤니케이션

- reactiveValues 활용

```r
# Module 1, which will allow to select a number
choice_ui <- function(id) {
  ns <- NS(id)
  tagList(
    # Add a slider to select a number
    sliderInput(ns("choice"), "Choice", 1, 10, 5)
  )
}

choice_server <- function(id, r) {
  moduleServer(
    id,
    function(input, output, session) {
      # Whenever the choice changes, the value inside r is set
      observeEvent( input$choice , {
       r$number_from_first_mod <- input$choice
      })

    }
  )
}
```

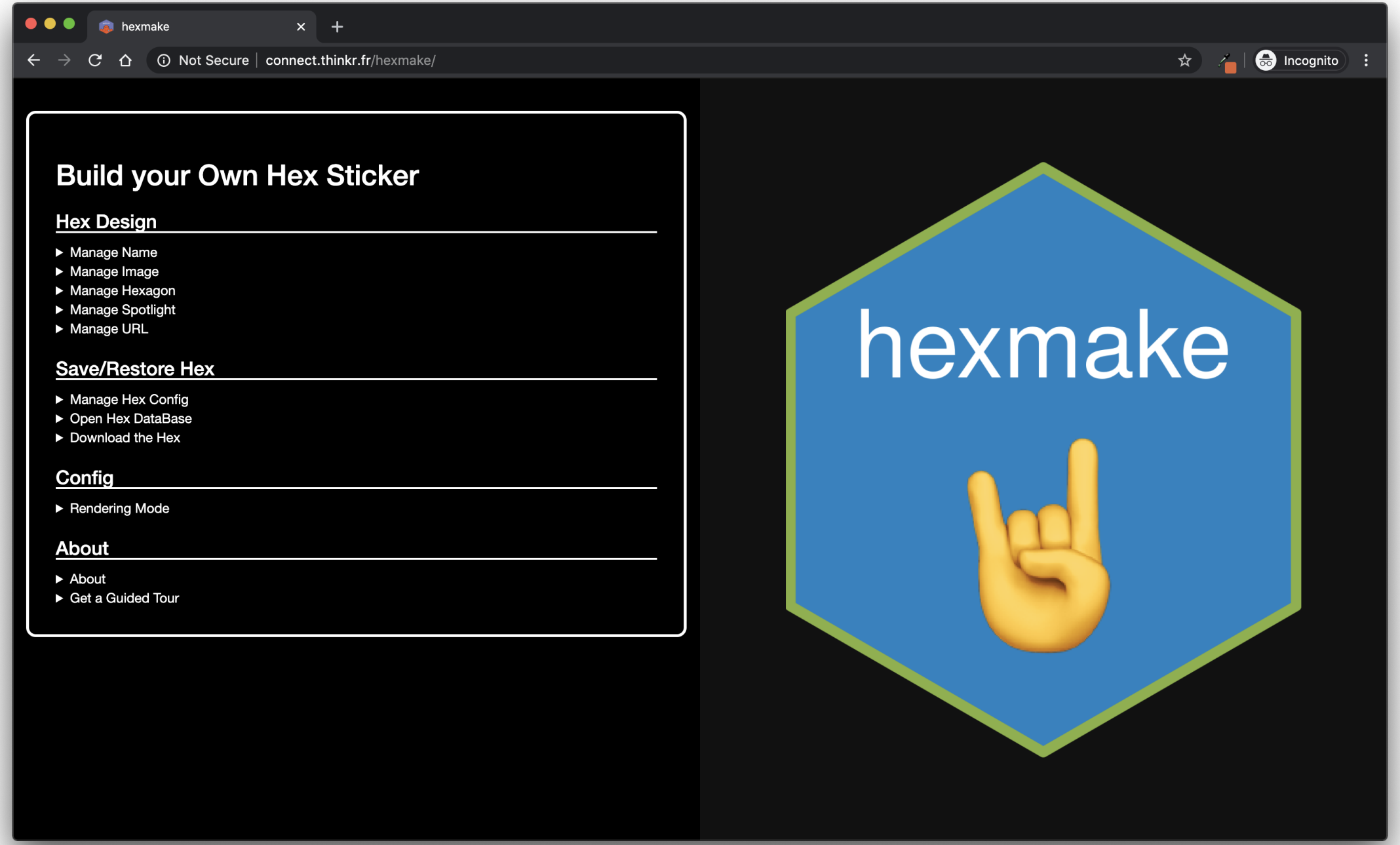
```r
# Module 2, which will display the number
printing_ui <- function(id) {
  ns <- NS(id)
  tagList(
    # Insert the number modified in the first module
    verbatimTextOutput(ns("print"))
  )
}

printing_server <- function(id, r) {
  moduleServer(
    id,
    function(input, output, session) {
      # We evaluate the reactiveValue element modified in the
      # first module
      output$print <- renderPrint({
       r$number_from_first_mod
      })
    }
  )
}
```

```r
# Application
library(shiny)
app_ui <- function() {
  fluidPage(
    choice_ui("choice_ui_1"),
    printing_ui("printing_ui_2")
  )
}

app_server <- function(input, output, session) {
  # both servers take a reactiveValue,
  #  which is set in the first module
  # and printed in the second one.
  # The server functions don't return any value per se
  r <- reactiveValues()
  choice_server("choice_ui_1", r = r)
  printing_server("printing_ui_2", r = r)
}

shinyApp(app_ui, app_server)
```

- 다른 옵션들: {R6}, {tidymodules}

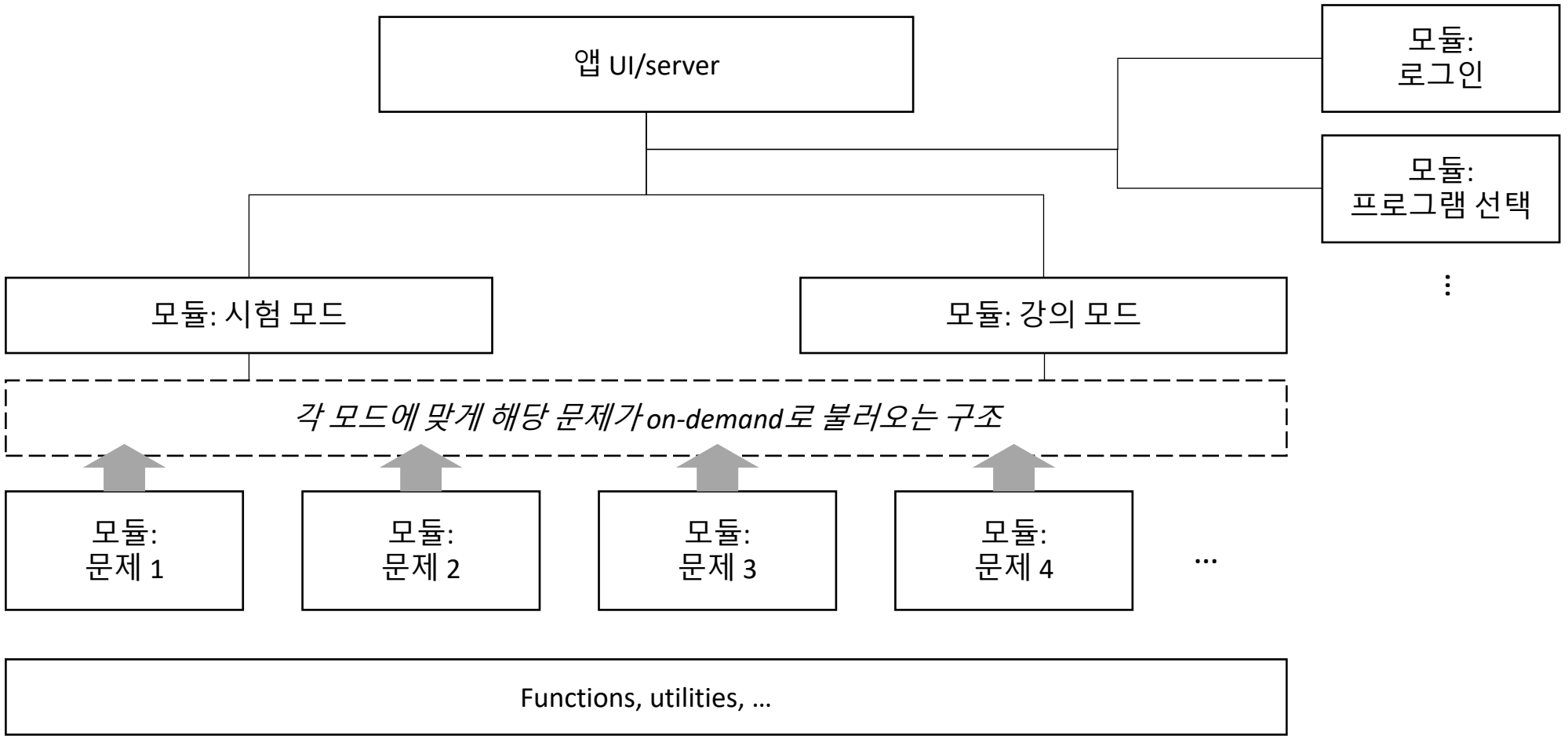
Source: https://engineering-shiny.org/structuring-project.html#communication-between-modules

Use cases



https://github.com/ColinFay/hexmake

Deep Skill 앱 (개발 중)



Miscellaneous

깨알 같은 utility들

```r
#' Columns wrappers
#'
#' These are convenient wrappers around
#' `column(12, ...)`, `column(6, ...)`, `column(4, ...)`
#'
#' @noRd
#'
#' @importFrom shiny column
col_12 <- function(...){
  column(12, ...)
}

#' @importFrom shiny column
col_10 <- function(...){
  column(10, ...)
}

#' @importFrom shiny column
col_8 <- function(...){
  column(8, ...)
}

#' @importFrom shiny column
col_6 <- function(...){
  column(6, ...)
}
```

```r
observeEvent(input$hidebutton1, {
  golem::invoke_js("hideid", "button1")
})

observeEvent(input$showbutton1, {
  golem::invoke_js("showid", "button1")
})
```

```
> golem::browser_button()
── To be copied in your UI ──────────────────────
actionButton("browser", "browser"),
tags$script("$('#browser').hide();")

── To be copied in your server ──────────────────
observeEvent(input$browser,{
  browser()
})

By default, this button will be hidden.
To show it, open your web browser JavaScript console
And run $('#browser').show();
```

End of Document