

Shiny Aide-mémoire

Plus d'infos sur shiny.rstudio.com

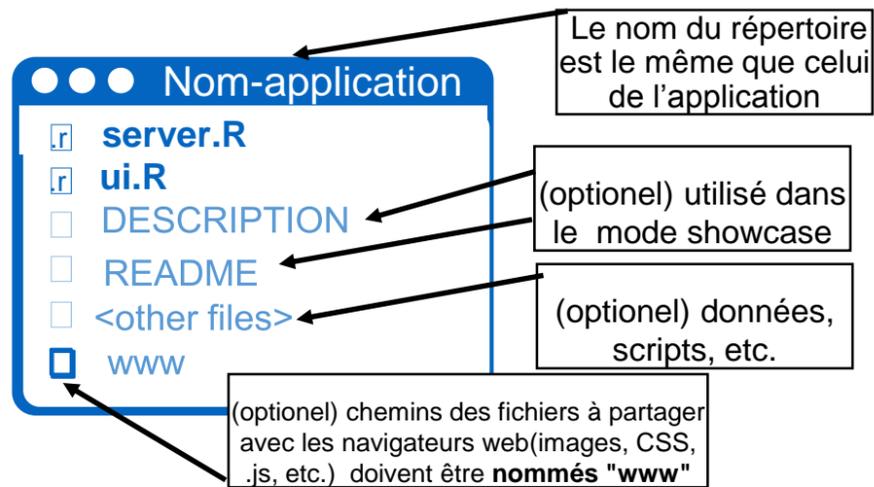
Shiny 0.10.0 Updated: 6/14



2. server.R est un ensemble d'instructions permettant de construire les éléments de l'application Shiny. Pour écrire le server.R:

- A** Ecrire server.R avec le minimum de code nécessaire, `shinyServer(function(input, output) {})`
- B** Définir les composantes R de l'application entre les accolades qui suivent `function(input, output)`
- C** Enregistrer chaque composante R dans l'UI comme `output$<nom composante>`
- D** Créer chaque composante `output` avec une fonction `render*`
- E** Donner à chaque fonction `render*` le code R nécessaire au `Server` pour construire la composante. Le serveur va reconnaître chaque valeur réactive qui apparaît dans le code et va la reconstruire chaque fois que sa valeur change
- F** Faire référence aux valeurs des widgets avec `input$<nom widget>`

1. Structure Chaque application est un répertoire contenant un fichier `server.R` et un fichier `ui.R` (et éventuellement des fichiers facultatifs)



Les fonctions `render*`

fonction	prend	crée
<code>renderDataTable</code>	tout objet équivalent à un tableau	<code>dataTables.js</code> table
<code>renderImage</code>	liste d'attributs d'image	image HTML
<code>renderPlot</code>	plot	plot
<code>renderPrint</code>	tout output imprimé	texte
<code>renderTable</code>	tout objet équivalent à un tableau	table
<code>renderText</code>	chaîne de caractères	texte
<code>renderUI</code>	objet Shiny tag ou HTML	élément UI(HTML)

Les valeurs d'`input` réactives doivent être utilisées dans :

- render*** - crée une composante UI
- reactive** - crée une expression réactive
- observe** - crée un **reactive observer**
- isolate** - crée une copie non réactive d'un objet réactif

server.R

```
# charger les librairies, scripts, et données
A shinyServer(function(input, output) { B
  # définir des variables spécifiques à l'utilisateur
  output$text <- renderText({
    input$title
  })
  C output$plot <- renderPlot({ D
    E x <- mtcars[ , input$x] F
    y <- mtcars[ , input$y]
    plot(x, y, pch = 16)
  })
})
```

3. Exécution Placer le code où il sera exécuté le minimum nécessaire de fois

- Exécuté une seule fois**- Le code placé à l'extérieur du `shinyServer` s'exécute une seule fois, lors du 1^{er} lancement de l'`app`. Utiliser ce code pour mettre en place les éléments dont le serveur n'a besoin qu'une seule fois.
- Exécuté une seule fois par utilisateur**- Le code placé dans le `shinyServer` va être exécuté chaque fois qu'un utilisateur lance l'`app` (ou rafraîchit son navigateur). Utiliser ce code pour mettre en place les éléments qui ne seront nécessaires qu'une fois pour chaque utilisateur,
- Exécuté souvent**- Le code placé dans une fonction `render*`, `reactive`, ou `observe` va être exécuté plusieurs fois. Placer ici uniquement le code dont le serveur a besoin pour reconstruire une composante UI après la modification d'un widget.

4. Réactivité (Quand un `input` change, le serveur va reconstruire chaque `output` qui en dépend(même quand la dépendance est indirecte) Ce comportement est maîtrisé par l'ajustement de la chaîne de dépendance.

render* - Un `output` sera automatiquement mis à jour quand un `input` de sa fonction `render*` change.



```
output$z <- renderText({
  input$a
})
```

Expression réactive - Utiliser `reactive` pour créer des objets à utiliser dans des multiples `outputs`.



```
x <- reactive({
  input$a
})
output$y <- renderText({
  x()
})
output$z <- renderText({
  x()
})
```

isolate - Utiliser `isolate` pour utiliser des `input` sans qu'il y ait de dépendance. Shiny ne reconstruit pas l'`output` quand l'`input` isolé change.



```
output$z <- renderText({
  paste(
    isolate(input$a),
    input$b
  )
})
```

observe - Utiliser `observe` pour le code exécuté quand un `input` change, mais sans créer d'`output`.



```
observe({
  input$a
  # code à exécuter
})
```

ui.R

A shinyUI(fluidPage(

```

titlePanel("Données mtcars"),
sidebarLayout(
  B sidebarPanel(
    C textInput("titre", "titre du graphique:",
      value = "x v y"),

    selectInput("x", "Choisissez une variable x:",
      choices = names(mtcars),
      selected = "disp"),

    selectInput("y", "Choisissez une variable y:",
      choices = names(mtcars),
      selected = "mpg")
  ),

  mainPanel(
    h3(textOutput("text")),
    plotOutput("plot")
  )
)
)

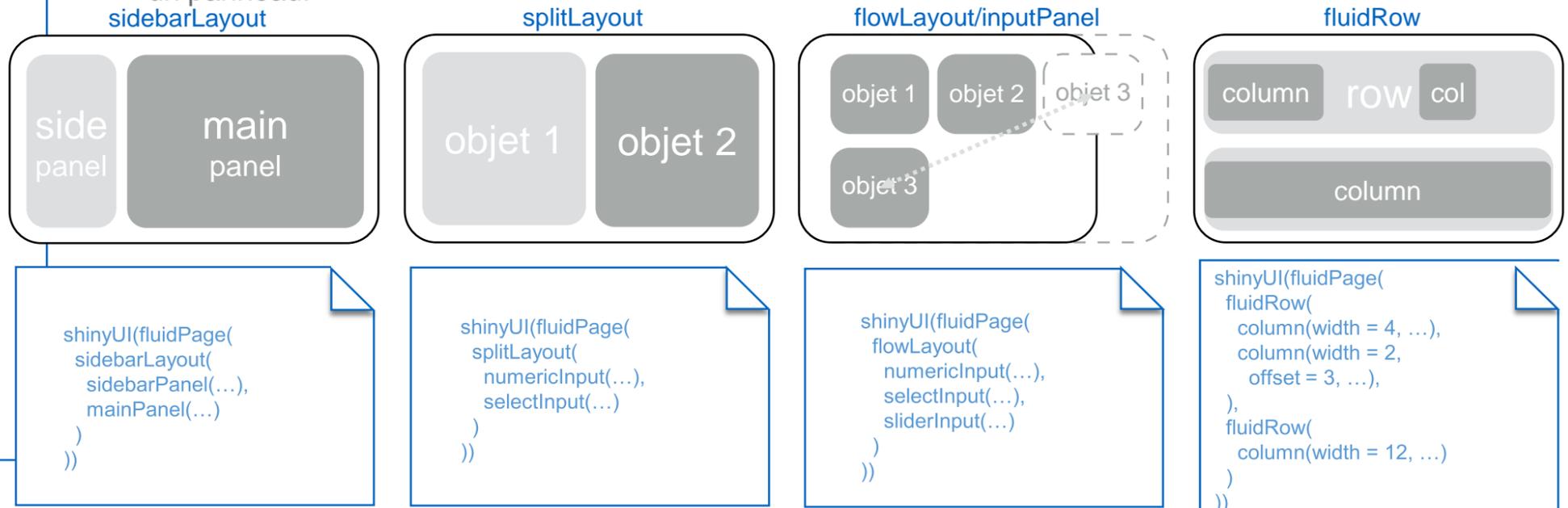
```

5. ui.R Une description de l'interface utilisateur (UI) de votre app, la page web qui affiche votre app. Pour écrire ui.R:

A Fournir le minimum nécessaire pour ui.R, shinyUI(fluidPage())

* note: utiliser navbarPage au lieu de fluidPage si l'app doit comprendre plusieurs pages connectées par une barre de navigation

B Construire un squelette pour l'UI. SidebarLayout fournit un squelette par défaut lorsqu'il est utilisé avec sidebarPanel et mainPanel. SplitLayout, flowLayout, et inputLayout divisent la page en régions équi-espacées. FluidRow et column fonctionnent de pair pour créer un squelette en grille, utilisable pour présenter une page ou un panneau.



C Dans chaque panneau ou colonne, placer:



Les composantes R - Ce sont les objets *output* définis dans server.R. Pour placer une composante:

1. Choisir une fonction **Output* qui construit le type d'objet à placer dans UI.
2. Passer à la fonction **Output* une chaîne de caractères qui correspond au nom de l'objet dans server.R, exemple:

```
output$plot <- renderPlot({ ... }) ↔ plotOutput("plot")
```

fonctions *Outputs

- | | |
|-----------------|--------------------|
| dataTableOutput | tableOutput |
| htmlOutput | textOutput |
| imageOutput | uiOutput |
| plotOutput | verbatimTextOutput |

Widgets - Le 1^{er} argument de chaque fonction widget est le nom pour le widget. Utiliser `input$<nom>` pour accéder à la valeur courante du widget dans server.R



widget	fonctions	arguments courants
Action button	actionButton	inputId, label
checkbox	checkboxInput	inputId, label, value
checkbox group	checkboxGroupInput	inputId, label, choices, selected
date selector	dateInput	inputId, label, value, min, max, format
date range selector	dateRangeInput	inputId, label, start, end, min, max, format
file uploader	fileInput	inputId, label, multiple
Number field	numericInput	inputId, label, value, min, max, step
Radio buttons	radioButtons	inputId, label, choices, selected
select box	selectInput	inputId, label, choices, selected, multiple
slider	sliderInput	inputId, label, min, max, value, step
submit button	submitButton	text
text field	textInput	inputId, label, value

HTML Eléments HTML - Ajout des éléments html avec les fonctions Shiny similaires aux tags HTML.

a	tags\$col	tags\$form	tags\$input	tags\$output	tags\$tbody
tags\$abbr	tags\$colgroup	h1	tags\$ins	p	tags\$thead
tags\$address	tags\$command	h2	tags\$kbd	tags\$param	tags\$tr
tags\$area	tags\$data	h3	tags\$keygen	pre	tags\$tbody
tags\$article	tags\$datalist	h4	tags\$label	tags\$progress	tags\$td
tags\$aside	tags\$dd	h5	tags\$legend	tags\$q	tags\$tfoot
tags\$audio	tags\$del	h6	tags\$li	tags\$ruby	tags\$tr
tags\$b	tags\$details	tags\$head	tags\$link	tags\$rp	tags\$tr
tags\$base	tags\$dfn	tags\$header	tags\$mark	tags\$rt	tags\$thead
tags\$bdi	div	tags\$hgroup	tags\$map	tags\$s	tags\$tbody
tags\$bdo	tags\$d	hr	tags\$menu	tags\$samp	tags\$tr
tags\$blockquote	tags\$dt	HTML	tags\$meta	tags\$script	tags\$tr
e	em	tags\$i	tags\$meter	tags\$section	tags\$tr
tags\$body	tags\$embed	tags\$iframe	tags\$nav	tags\$select	tags\$tbody
br	tags\$eventsource	img	tags\$noscript	tags\$small	tags\$thead
tags\$button	ce	includeCSS	tags\$object	tags\$span	tags\$tbody
tags\$canvas	tags\$fieldset	includeMarkd	tags\$ol	tags\$source	tags\$tbody
tags\$caption	tags\$figcaption	own	tags\$optgroup	span	tags\$tbody
tags\$cite	tags\$figure	includeScript	tags\$option	strong	tags\$tbody
code	tags\$footer			tags\$style	

6. Exécuter votre app

- runApp** - exécuter en local
- runGitHub** - exécuter depuis des fichiers hébergés sur www.GitHub.com
- runGist** - exécuter depuis des fichiers enregistrés comme un gist (gist.github.com)
- runURL** - exécuter depuis des fichiers enregistrés dans des URL

7. Partager votre app Lancer votre app comme une page web dynamique que les utilisateurs peuvent consulter en ligne

ShinyApps.io

Héberger votre app sur le serveur RStudio. Options gratuites et payantes.
www.shinyapps.io

Shiny Server

Construire un serveur linux pour héberger votre app. Gratuit et open source.
shiny.rstudio.com/deploy

Shiny Server Pro

Construire un serveur commercial avec authentification, gestion des ressources, et plus.
shiny.rstudio.com/deploy