

Réaliser une appli avec {shiny} & {bslib}

Marie Vaugoyeau

16 December 2025

Sommaire

1	S'initier à la création d'applications web	2
1.1	C'est quoi une appli web ?	2
1.2	Construire une application	2
1.3	Shiny	2
1.4	Explorer Shiny	3
1.5	Exemples de réalisation	3
1.6	Comprendre la structure d'une application	3
1.7	Création d'une structure de page avec {bslib}	4
2	Application développée lors du live	9
2.1	Une appli sur les pingouins	9
2.2	Initier la structure	9
2.3	Créer la structure	9
2.4	Personnaliser l'interface	11
2.5	Ajouter un élément interactif	13
2.6	Paramétrer les choix de l'utilisateurice	16
2.7	Communiquer avec l'utilisateurice	21

Note

[Twitch du 16 décembre 2025.](#)

1 S'initier à la création d'applications web

1.1 C'est quoi une appli web ?

- Une applications web est un objet informatique **manipulable directement, en ligne et sans installation**
- Une application peut aussi être présente sur le réseau interne d'une entreprise, sur un unique ordinateur ou hébergée en ligne.

1.2 Construire une application

- Avoir une idée
- Documenter les besoins et attentes des utilisateur.trice.s
- Relier les besoins et attentes à des actions
- Lister les données nécessaires puis faire **valider**
- Réaliser une maquette simple et la **valider**
- Construire une maquette fidèle et la **valider**
- Coder l'application
- Faire tourner en local
- Déployer l'application
- Faire évoluer ou abandonner

1.3 Shiny

`{shiny}` est un package R développé par  **posit** en 2012.



1.4 Explorer Shiny

- Avant Shiny, il était nécessaire d'avoir une bonne connaissance des technologies du web (HTML, CSS,...)
- Shiny permet de :
 - générer simplement le front-end (l'interface utilisateur.trice) et le back-end sans apprendre de langage web
 - faire des tableaux de bords mais aussi des pages web pour communiquer les résultats
 - de travailler directement sur les données de la personne utilisatrice
 - de gérer des cartes, des tableaux, des graphiques, des questionnaires...

1.5 Exemples de réalisation

Beaucoup d'exemples sont disponibles dans la [galerie de Shiny](#)

1.6 Comprendre la structure d'une application

Toutes les applications ont :

- une partie **User Interface** qui permet de gérer le **Front-End**, c'est-à-dire à quoi l'**application** ressemble
- une partie **Server** qui gère le **Back-End**, c'est-à-dire les mécanismes sous-jacents ou **que fait l'application**

i A retenir

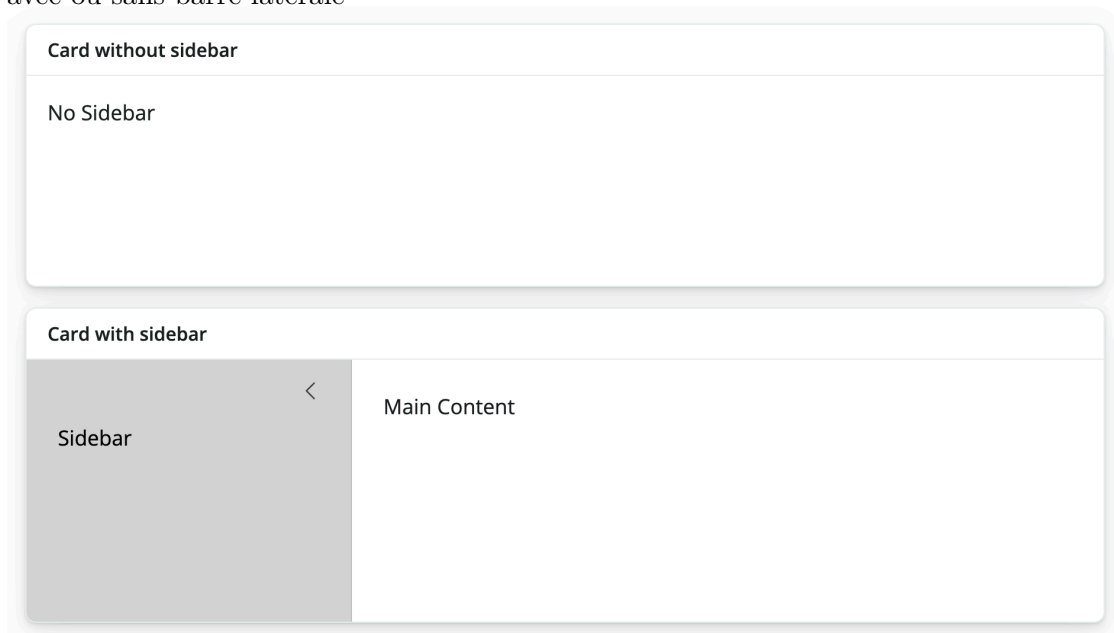
Dans une application **Shiny**, l'interface utilisateur.trice est géré par la fonction ou le fichier **ui** et les coulisses par la fonction ou le fichier **Server**.

1.7 Création d'une structure de page avec {bslib}

Avec le package **{bslib}**, une page est construite à partir d'une ou plusieurs sections appelées **card**

Les **card** peuvent-être :

- avec ou sans barre latérale



```
library(bslib)

ui <- page_fillable(

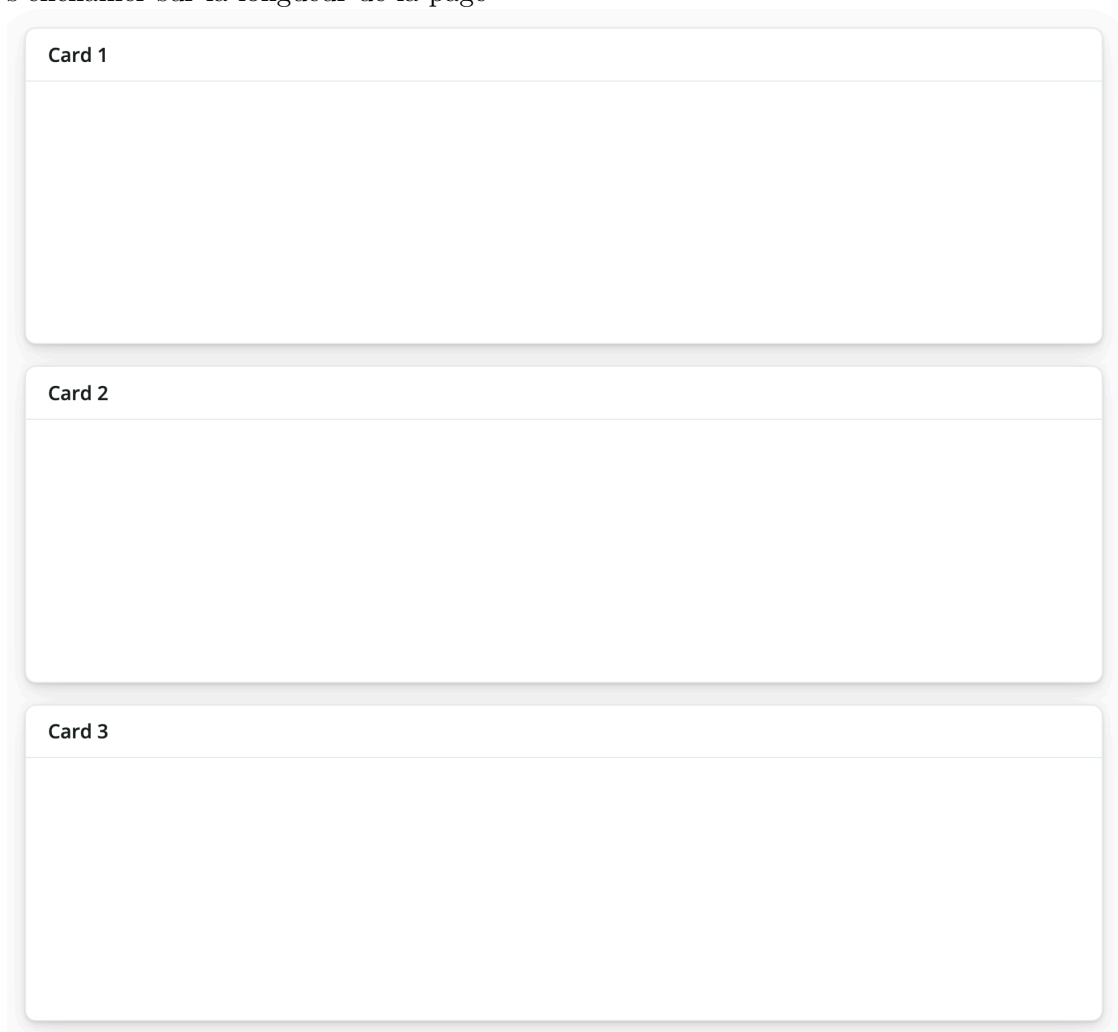
  card(
    card_header("Card without sidebar"),
    "No Sidebar"
  ),
```

```

card(
  card_header("Card with sidebar"),
  layout_sidebar(
    sidebar = sidebar(
      bg = "lightgrey",
      "Sidebar"
    ),
    "Main Content"
  )
)
)

```

- s'enchaîner sur la longueur de la page



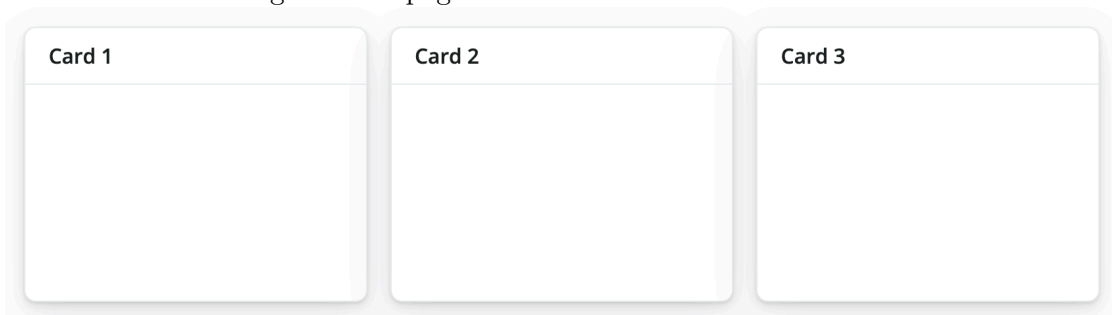
```
library(bslib)

ui <- page_fillable(

  card(card_header("Card 1")),
  card(card_header("Card 2")),
  card(card_header("Card 3"))

)
```

- s'enchaîner sur la largeur de la page



```
library(bslib)

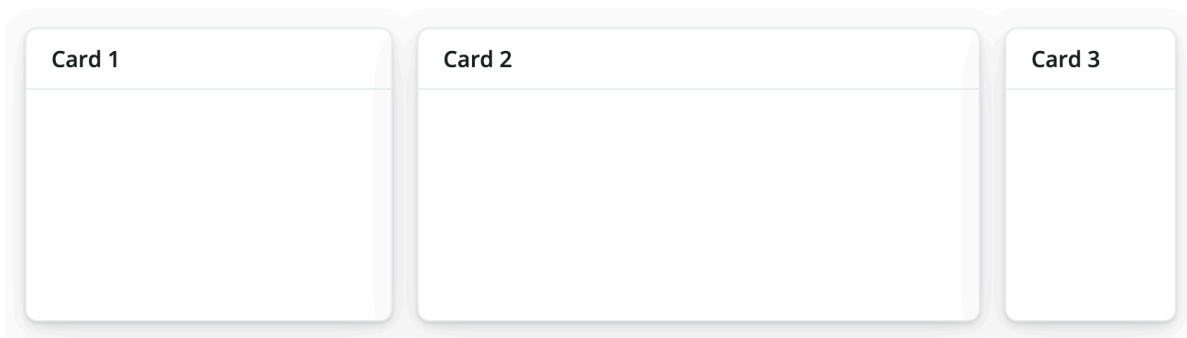
ui <- page_fillable(

  layout_columns(
    card(card_header("Card 1")),
    card(card_header("Card 2")),
    card(card_header("Card 3"))
  )

)
```

💡 Modifier la largeur et hauteur des cards

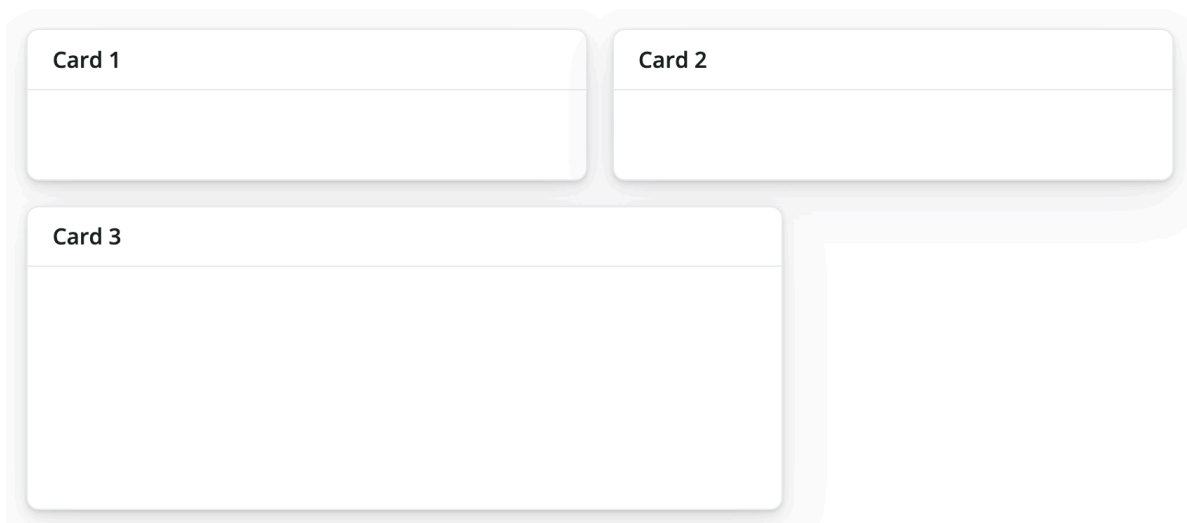
L'argument `col_widths` permet de paramétrer la largeur de chaque `card`.
 Si la somme d'une seule ligne est supérieure à 12, une nouvelle ligne est créée par défaut.
 L'argument `row_heights` permet de fixer le rapport entre les lignes (une valeur fixe peut éventuellement être utilisée).



```
library(bslib)

ui <- page_fillable(

  layout_columns(
    card(card_header("Card 1")),
    card(card_header("Card 2")),
    card(card_header("Card 3")),
    col_widths = c(4, 6, 2)
  )
)
```



```
library(bslib)

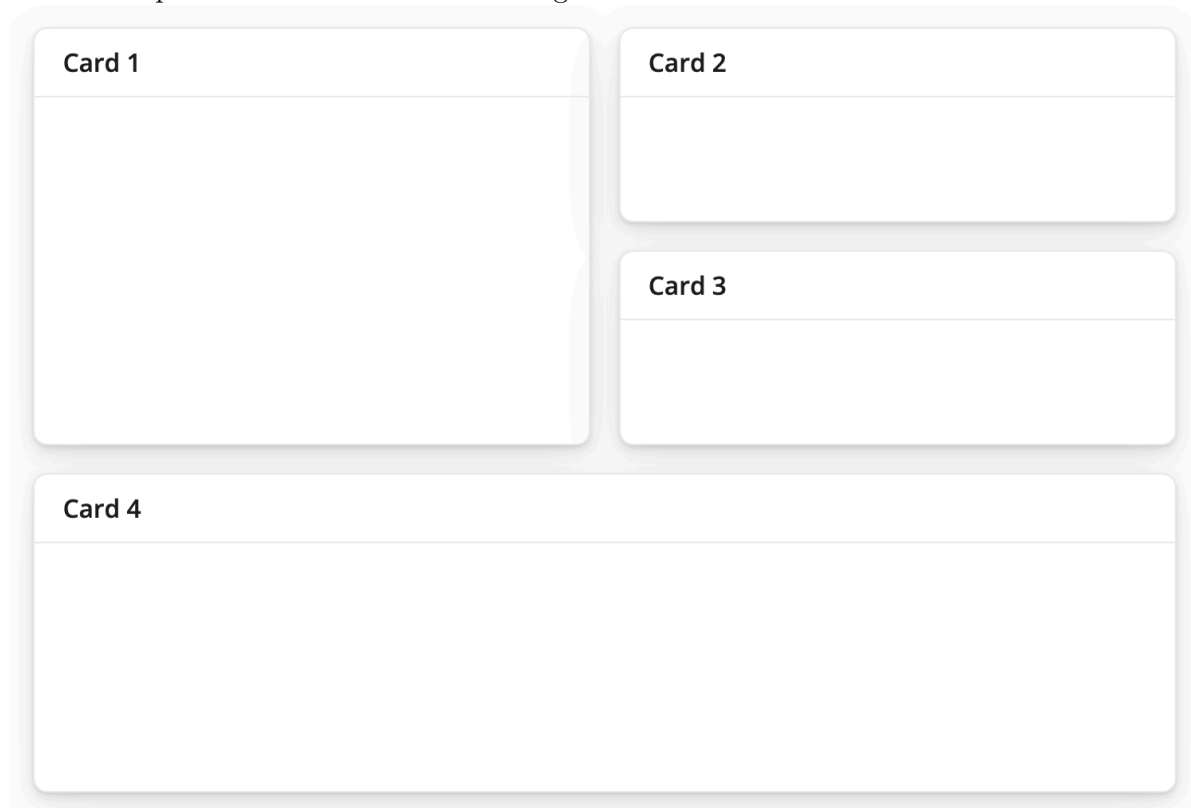
ui <- page_fillable(
```

```

layout_columns(
  card(card_header("Card 1")),
  card(card_header("Card 2")),
  card(card_header("Card 3")),
  col_widths = c(6, 6, 8),
  row_heights = c(1, 2)
)
)

```

Il est aussi possible de mixer les `card` en lignes et colonnes



```

library(bslib)

ui <- page_fillable(

  layout_columns(
    card(card_header("Card 1")),
    layout_columns(

```

```

    card(card_header("Card 2")),
    card(card_header("Card 3")),
    col_widths = c(12, 12)
  )
),
  card(card_header("Card 4"))
)

```

💡 Pour aller plus loin

[Application layout guide](#) sur le site de Shiny.

2 Application développée lors du live

2.1 Une appli sur les pingouins

Lors du live, j'ai développée [cette application](#), voici les différentes étapes

2.2 Initier la structure

Pour créer une appli Shiny, un seul fichier `.R` suffit.

💡 squelette d'une appli

Pour générer facilement le squelette d'une appli, taper `shinyapp` puis taper sur **Entrée** pour utiliser le `snippet`.

Afin de créer l'application, je vais appeler le package `{bslib}` et `{tidyverse}` en plus du package `{shiny}`.

2.3 Créer la structure

La structure choisie est un page avec une barre de navigation contenant le titre et deux onglets. Le premier onglet a dans la fenêtre une ligne avec deux colonnes et une deuxième ligne sur la totalité.

Le deuxième onglet contient du texte.

```

library(shiny)
library(bslib)
library(tidyverse)

ui <- page_navbar(
  title = "Les penguins",
  nav_panel(
    "Représentations",
    layout_columns(
      card(
        full_screen = TRUE,
        p("graphique sur la forme du bec")
      ),
      card(
        full_screen = TRUE,
        p("graphique sur la condition coporelle")
      )
    ),
    card(
      full_screen = TRUE,
      card_header("Données"),
      p("tableau des données")
    )
  ),
  nav_panel(
    "Information",
    p("Cette appli a été créée dans le cadre de la formation R-Shiny"),
    p("Pour plus d'information contacter Marie (marie.vaugoyea@gmail.com)")
  )
)

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

}

shinyApp(ui, server)

```

graphique sur la forme du bec

graphique sur la condition corporelle

Données

tableau des données

2.4 Personnaliser l'interface

En ajoutant un thème grâce à la fonction `bs_theme()` du package `{bslib}` : `theme = bs_theme(bootswatch = "morph")`.

La ligne doit être ajoutée dans la fonction `page_navbar()`. Je mets généralement **en premier** mais ce n'est pas obligatoire.

En ajoutant des barres latérales afin de donner la possibilité de choisir une espèce pour le tableau et une île pour le graphique sur la condition corporelle.

Le fond des barres latérales a été modifiés grâce à l'argument `bg`.

```
library(shiny)
library(bslib)
library(tidyverse)
```

```

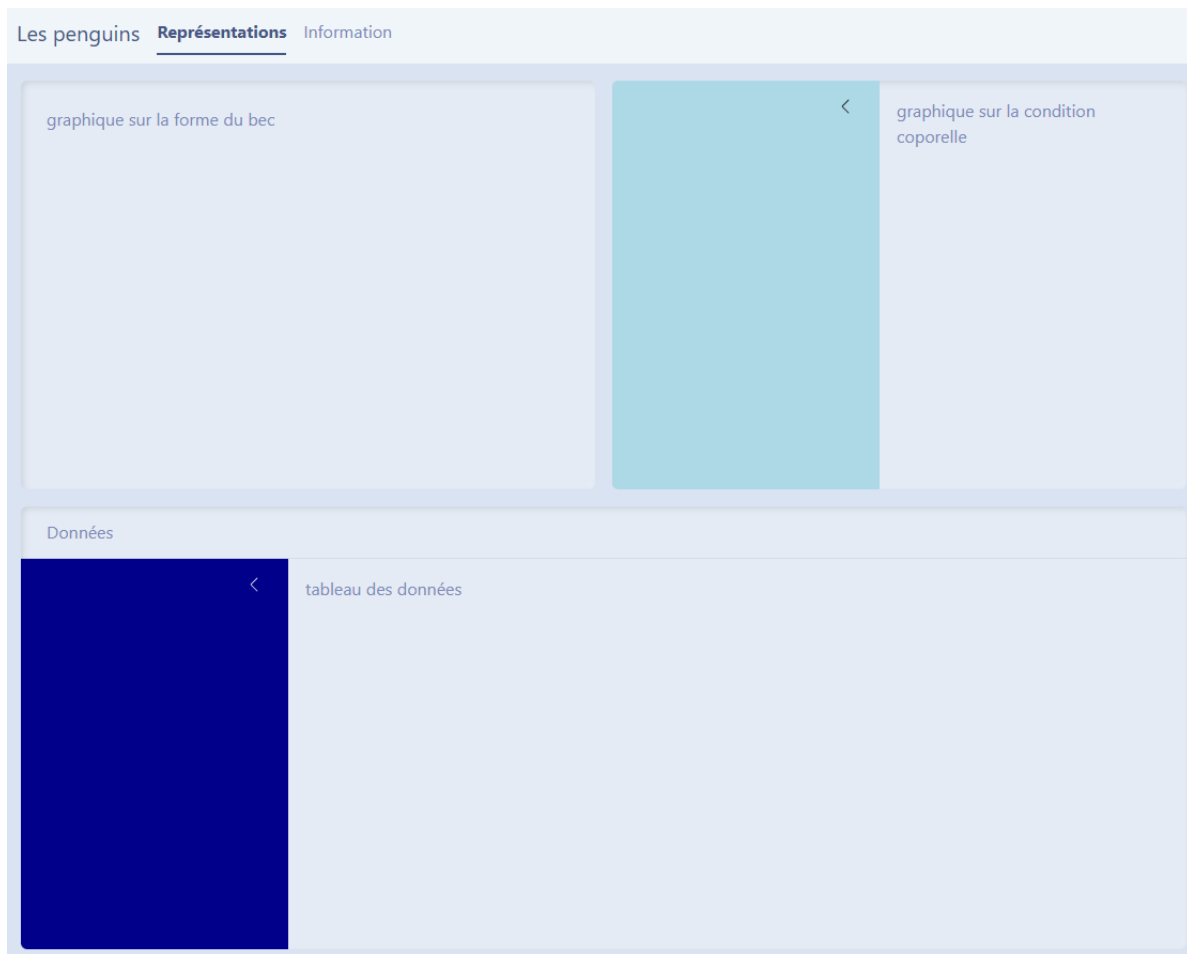
ui <- page_navbar(
  theme = bs_theme(bootswatch = "morph"),
  title = "Les penguins",
  nav_panel(
    "Représentations",
    layout_columns(
      card(
        full_screen = TRUE,
        p("graphique sur la forme du bec")
      ),
      card(
        full_screen = TRUE,
        layout_sidebar(
          sidebar = sidebar(
            bg = "lightblue"
          ),
          p("graphique sur la condition coporelle")
        )
      )
    ),
    card(
      full_screen = TRUE,
      card_header("Données"),
      layout_sidebar(
        sidebar = sidebar(
          bg = "darkblue"
        ),
        p("tableau des données")
      )
    )
  ),
  nav_panel(
    "Information",
    p("Cette appli a été créée dans le cadre de la formation R-Shiny"),
    p("Pour plus d'information contacter Marie (marie.vaugoyea@gmail.com)")
  )
)

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

}

```

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



2.5 Ajouter un élément interactif

Le premier graphique ajouté est celui de la forme du bec. Il est rendu interactif par l'utilisation du package `{plotly}`.

! interactif = server au repos

L'interactivité repose sur l'idée que l'utilisateur peut influencer le comportement de l'application sans relancer le code.

```

library(shiny)
library(bslib)
library(tidyverse)
library(plotly)

# creation vecteur_couleur
vecteur_quali <-
  c(
    "Adelie" = "#DF9ED4FF",
    "Chinstrap" = "#C93F55FF",
    "Gentoo" = "#EACC62FF",
    "Biscoe" = "#469D76FF",
    "Dream" = "#3C4B99FF",
    "Torgersen" = "#924099FF"
  )

ui <- page_navbar(
  theme = bs_theme(bootswatch = "morph"),
  title = "Les penguins",
  nav_panel(
    "Représentations",
    layout_columns(
      card(
        full_screen = TRUE,
        plotlyOutput("bec")
      ),
      card(
        full_screen = TRUE,
        layout_sidebar(
          sidebar = sidebar(
            bg = "lightblue"
          ),
          p("graphique sur la condition coporelle")
        )
      )
    ),
    card(
      full_screen = TRUE,
      card_header("Données"),
      layout_sidebar(
        sidebar = sidebar(
          bg = "darkblue"
        )
      )
    )
  )
)

```

```

    ),
    p("tableau des données")
  )
),
nav_panel(
  "Information",
  p("Cette appli a été créée dans le cadre de la formation R-Shiny"),
  p("Pour plus d'information contacter Marie (marie.vaugoyea@gmail.com)")
)
)

server <- function(input, output, session) {
  output$bec <- renderPlotly(
    penguins |>
      drop_na() |>
      ggplot() +
      aes(
        x = bill_dep,
        y = bill_len,
        shape = sex,
        color = species,
        linetype = sex
      ) +
      geom_point() +
      geom_smooth(method = "lm", se = FALSE) +
      scale_color_manual(values = vecteur_quali) +
      scale_shape_manual(values = c(17,6)) +
      labs(
        title = "Forme du bec",
        x = "largeur du bec (mm)",
        y = "longueur du bec (mm)",
        color = "espèce",
        shape = "sexe",
        linetype = "sexe"
      ) +
      theme_minimal()
  )
}

shinyApp(ui, server)

```



Tu peux voir ici qu'un objet `bec` a été créé dans la partie **Server** grâce à `renderPlotly()` du package `{plotly}` et appelé dans la partie **ui** grâce à la fonction `plotlyOutput()` du même package.

2.6 Paramétrer les choix de l'utilisatrice

Pour le graphique, la personne pourra choisir l'île ou les îles à représenter et dans le tableau, le choix de(s) espèce(s).

Dans ce cas, il faut ajouter deux **widgets** qui va permettre de rendre réactif l'appli shiny.

! réactivité = code tourne

La réactivité repose sur l'idée que l'application réagit dynamiquement et sans délai aux actions de l'utilisateur.trice.

Chaque choix est enregistré dans un `input` qui a un nom unique puis utilisé dans la partie `server`.

```
library(shiny)
library(bslib)
library(tidyverse)
library(plotly)
library(DT)

# creation vecteur_couleur
vecteur_quali <-
  c(
    "Adelie" = "#DF9ED4FF",
    "Chinstrap" = "#C93F55FF",
    "Gentoo" = "#EACC62FF",
    "Biscoe" = "#469D76FF",
    "Dream" = "#3C4B99FF",
    "Torgersen" = "#924099FF"
  )

ui <- page_navbar(
  theme = bs_theme(bootswatch = "morph"),
  title = "Les penguins",
  nav_panel(
    "Représentations",
    layout_columns(
      card(
        full_screen = TRUE,
        plotlyOutput("bec")
      ),
      card(
        full_screen = TRUE,
        layout_sidebar(
          sidebar = sidebar(
            bg = "lightblue",
            checkboxGroupInput(
              "ile",
              "Choisissez une île",
              selected = penguins$island |> fct_unique(),
              choiceNames = penguins$island |> fct_unique(),
              choiceValues = penguins$island |> fct_unique()
            )
          )
        )
      )
    )
  )
),
```

```

        plotOutput("masse")
      )
    )
  ),
  card(
    full_screen = TRUE,
    card_header("Données"),
    layout_sidebar(
      sidebar = sidebar(
        bg = "darkblue",
        checkboxGroupInput(
          "espece",
          "Choisissez une espèce",
          selected = penguins$species |> fct_unique(),
          choiceNames = penguins$species |> fct_unique(),
          choiceValues = penguins$species |> fct_unique()
        )
      ),
      DTOutput("table")
    )
  ),
  nav_panel(
    "Information",
    p("Cette appli a été créée dans le cadre de la formation R-Shiny"),
    p("Pour plus d'information contacter Marie (marie.vaugoyea@gmail.com)")
  )
)

server <- function(input, output, session) {
  output$bec <- renderPlotly(
    penguins |>
      drop_na() |>
      ggplot() +
      aes(
        x = bill_dep,
        y = bill_len,
        shape = sex,
        color = species,
        linetype = sex
      ) +
      geom_point() +

```

```

    geom_smooth(method = "lm", se = FALSE) +
    scale_color_manual(values = vecteur_quali) +
    scale_shape_manual(values = c(17,6)) +
    labs(
      title = "Forme du bec",
      x = "largeur du bec (mm)",
      y = "longueur du bec (mm)",
      color = "espèce",
      shape = "sexe",
      linetype = "sexe"
    ) +
    theme_minimal()
  )

output$masse <- renderPlot({
  penguins |>
    filter(island %in% input$file) |>
    drop_na() |>
    ggplot() +
    aes(
      x = body_mass,
      y = flipper_len,
      color = island
    ) +
    geom_point() +
    geom_smooth(method = "lm", se = FALSE) +
    scale_color_manual(values = vecteur_quali) +
    labs(
      title = "Condition corporelle",
      x = "masse (g)",
      y = "longueur de la nageoire (mm)",
      color = "île"
    ) +
    facet_wrap(~ species) +
    theme_minimal()
})

data <- reactive({
  penguins |> filter(species %in% input$espece)
})

output$table <- renderDT(

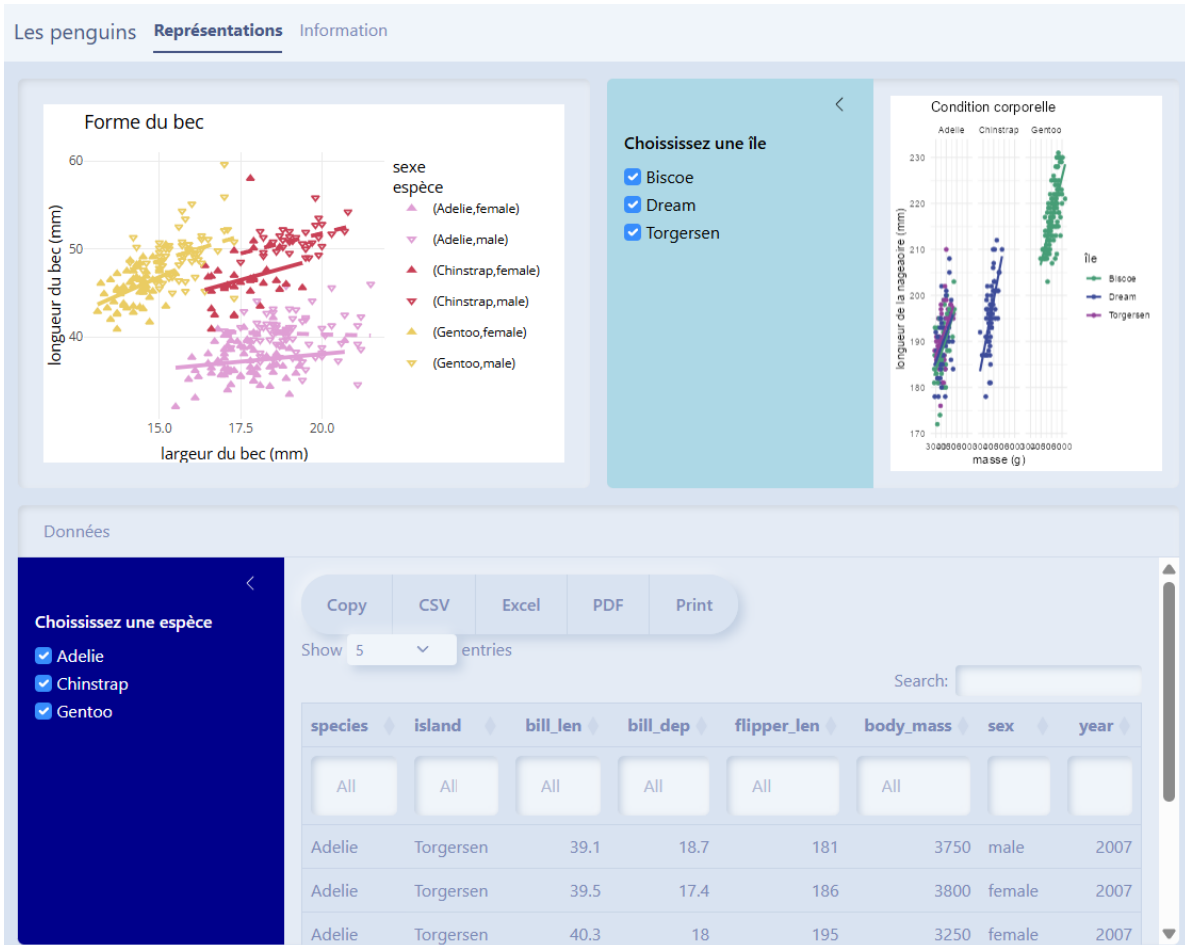
```

```

data(),
class = "cell-border compact",
extensions = c("Buttons"),
filter = "top",
options =
  list(
    pageLength = 5,
    dom = "Blftip",
    buttons = c(
      "copy", "csv", "excel", "pdf", "print"
    )
  ),
editable = "cell",
rownames = FALSE
)
}

shinyApp(ui, server)

```



Comme précédemment, les objets créés dans la partie **server** grâce aux fonctions **render*()** du package **{shiny}** et **{DT}** sont appelés dans la partie **ui** grâce aux fonctions miroirs ***Output()** des mêmes packages.

💡 **reactive()** de **{shiny}**

La fonction **reactive()** permet de créer un objet qui va être modifier à chaque modification de l'input associé.

2.7 Communiquer avec l'utilisatrice

Il est important de communiquer avec la(les) personne(s) utilisant l'appli à travers des messages pour la guider dans son utilisation.

Par exemple, ici, le choix d'au moins une espèce ou d'une île est obligatoire.

Dans ce cas, on va bloquer la création des sorties au fait de faire un choix grâce à la fonction `req()` du package `{shiny}`.

On va aussi envoyer un message en cas d'absence de choix grâce aux fonctions `textOutput()` et `renderText()` du package `{shiny}`.

```
library(shiny)
library(bslib)
library(tidyverse)
library(plotly)
library(DT)

# creation vecteur_couleur
vecteur_quali <-
  c(
    "Adelie" = "#DF9ED4FF",
    "Chinstrap" = "#C93F55FF",
    "Gentoo" = "#EACC62FF",
    "Biscoe" = "#469D76FF",
    "Dream" = "#3C4B99FF",
    "Torgersen" = "#924099FF"
  )

ui <- page_navbar(
  theme = bs_theme(bootswatch = "morph"),
  title = "Les penguins",
  nav_panel(
    "Représentations",
    layout_columns(
      card(
        full_screen = TRUE,
        plotlyOutput("bec")
      ),
      card(
        full_screen = TRUE,
        layout_sidebar(
          sidebar = sidebar(
            bg = "lightblue",
            checkboxGroupInput(
              "ile",
              "Choisissez une île",
              selected = penguins$island |> fct_unique(),
              choiceNames = penguins$island |> fct_unique(),
              choiceValues = penguins$island |> fct_unique()
            )
          )
        )
      )
    )
  )
)
```

```

    )
  ),
  plotOutput("masse"),
  textOutput("message_ile")
)
),
card(
  full_screen = TRUE,
  card_header("Données"),
  layout_sidebar(
    sidebar = sidebar(
      bg = "darkblue",
      checkboxGroupInput(
        "espece",
        "Choisissez une espèce",
        selected = penguins$species |> fct_unique(),
        choiceNames = penguins$species |> fct_unique(),
        choiceValues = penguins$species |> fct_unique()
      )
    ),
    DTOutput("table"),
    textOutput("message_espece")
  )
),
nav_panel(
  "Information",
  p("Cette appli a été créée dans le cadre de la formation R-Shiny"),
  p("Pour plus d'information contacter Marie (marie.vaugoyea@gmail.com)")
)
)

server <- function(input, output, session) {
  output$bec <- renderPlotly(
    penguins |>
      drop_na() |>
      ggplot() +
      aes(
        x = bill_dep,
        y = bill_len,
        shape = sex,

```

```

    color = species,
    linetype = sex
  ) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  scale_color_manual(values = vecteur_quali) +
  scale_shape_manual(values = c(17,6)) +
  labs(
    title = "Forme du bec",
    x = "largeur du bec (mm)",
    y = "longueur du bec (mm)",
    color = "espèce",
    shape = "sexe",
    linetype = "sexe"
  ) +
  theme_minimal()
)

output$masse <- renderPlot({
  req(input$file)
  penguins |>
    filter(island %in% input$file) |>
    drop_na() |>
    ggplot() +
    aes(
      x = body_mass,
      y = flipper_len,
      color = island
    ) +
    geom_point() +
    geom_smooth(method = "lm", se = FALSE) +
    scale_color_manual(values = vecteur_quali) +
    labs(
      title = "Condition corporelle",
      x = "masse (g)",
      y = "longueur de la nageoire (mm)",
      color = "île"
    ) +
    facet_wrap(~ species) +
    theme_minimal()
})

```

```

data <- reactive({
  req(input$espece)
  penguins |> filter(species %in% input$espece)
})

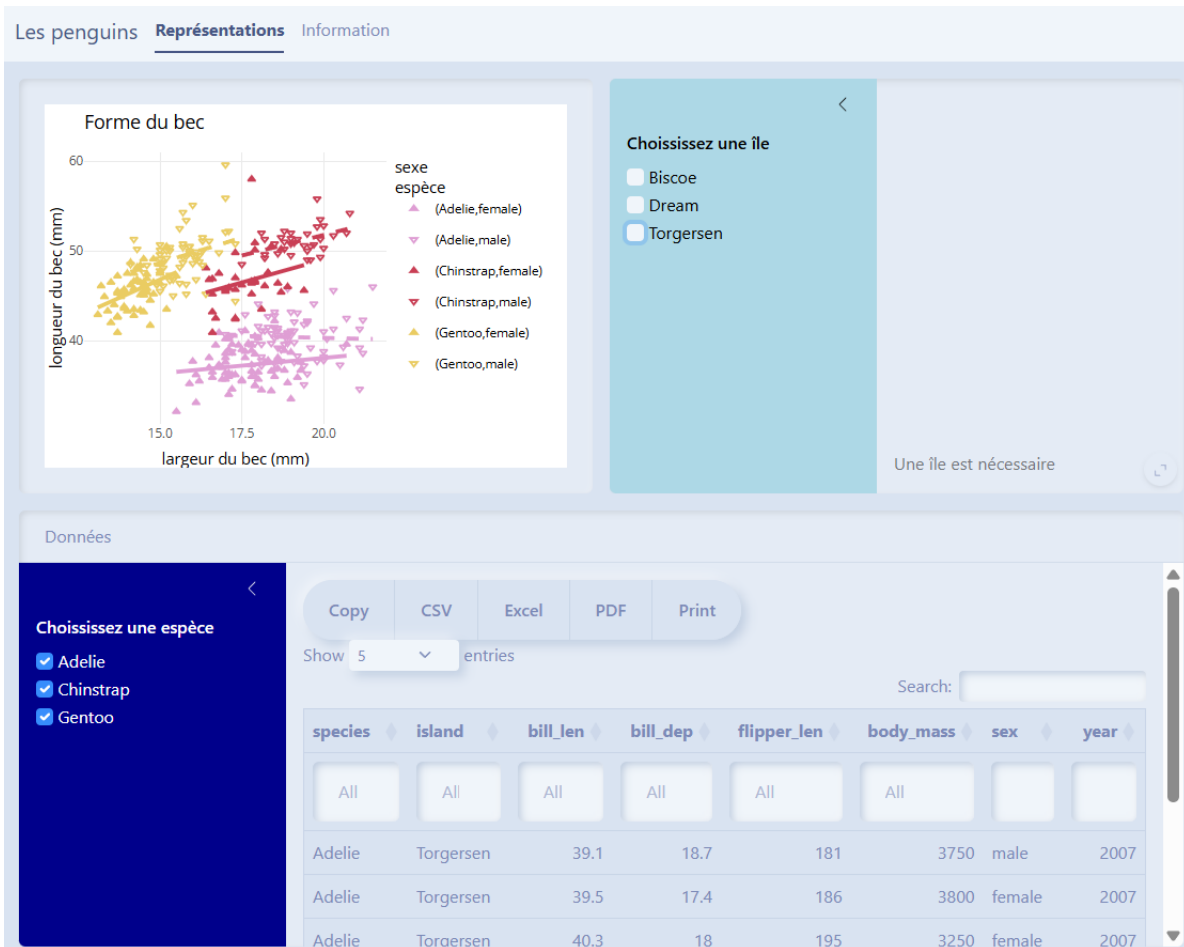
output$message_ile <- renderText(
  if (is.null(input$file)){validate("Une île est nécessaire")}
)

output$message_espece <- renderText(
  if (is.null(input$espece)){validate("Une espèce est nécessaire")}
)

output$table <- renderDT(
  data(),
  class = "cell-border compact",
  extensions = c("Buttons"),
  filter = "top",
  options =
    list(
      pageLength = 5,
      dom = "Blftip",
      buttons = c(
        "copy", "csv", "excel", "pdf", "print"
      )
    ),
  editable = "cell",
  rownames = FALSE
)
}

shinyApp(ui, server)

```



Et voilà, n'hésites pas à me faire des retours par [email](#) ou sur [LinkedIn](#)

i Note

Tu peux aussi t'inscrire à la [newsletter](#) pour être au courant du prochain live

Bonne journée