

GUIDE UTILISATEUR

DU DASHBOARD « INDICATEURS DE LA BIODIVERSITÉ AURA »

CO-AUTEURS :

Tomas Bonilla
Thomas Bultingaire
Laurianna Ferra
Thomas Logeais

Sommaire

Sommaire	2
Introduction	3
Aperçu de l'application web	3
Architecture des fichiers sources	3
Logique entre les dossiers et les fichiers sources	3
Visualisation de l'application	6
Récapitulatif des interactions utilisateurs	8
Page d'accueil & navigation	8
Page globale et experte	10
Ensemble des sélecteurs	13
Récapitulatif des variables de stockage de données	17
La maintenance des fichiers sources	18
Maintenir une information dans la page accueil	18
Ajouter une information générale sans catégorie d'informations	19
Ajouter une catégorie d'informations	21
Maintenir un type d'indicateur	24
Ajouter un type d'indicateur dans les sélecteurs	24
Ajouter un type d'indicateur dans la page d'accueil	24
Maintenir un indicateur	25
Ajouter un indicateur dans les sélecteurs	25
Ajouter une représentation d'un indicateur	25
Ajouter un indicateur dans la page d'accueil	26
Maintenir un mode de représentation	26
Maintenir une déclinaison et ses groupes associés	29
Ajouter une déclinaison dans les sélecteurs	29
Ajouter une taxonomie dans les sélecteurs	30
Réimplémentation de la page administrateur	31
Annexes	33

Introduction

Le dashboard « INDICATEURS DE LA BIODIVERSITÉ AURA » est un projet mis en place par l'Observatoire de la Biodiversité Auvergne-Rhône-Alpes et développé par une équipe d'étudiants en ingénierie du numérique de Télécom Saint-Etienne.

Ce projet a pour but de partager une visualisation claire, ergonomique, esthétique et adaptée aux profils des utilisateurs des indicateurs SRADDET sur l'état et l'évolution de la biodiversité en Auvergne-Rhône-Alpes.

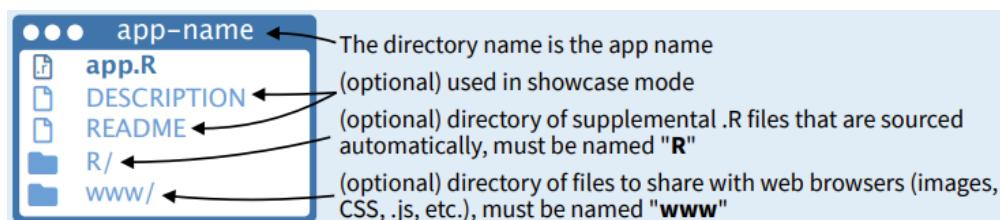
Afin de pouvoir assurer une maintenance et une évolution de l'application web, ce guide utilisateur retracera l'architecture de ses fichiers sources et la logique qu'elle implémente en se basant sur le langage R et sa bibliothèque shiny, puis énoncera les différents cas de figure où le mainteneur pourraient chercher à étendre l'application via des rajouts de morceaux de code.

Toutefois, ce guide ne vous expliquera pas jusque dans le moindre détail le fonctionnement de l'application. Pour plus d'informations particulières, n'hésitez pas à vous plonger dans le code source et à lire les commentaires.

I. Aperçu de l'application web

A. Architecture des fichiers sources

L'architecture des fichiers sources a été pensée afin de correspondre au modèle Shiny App qu'impose la bibliothèque Shiny.



Description générale de l'architecture d'une application R Shiny

De ce fait, l'architecture (*cf. Annexe Figure 1*), met en place un fichier *app.R* qui lance le serveur et deux dossiers, *R* et *www*. Le dossier *R* recense les scripts R que le serveur va importer et le dossier *www* les ressources web tels que les images et les scripts web. Ainsi, une première étape dans la maintenance de cette application web est de s'assurer que le code source est bien répartie dans l'architecture.

B. Logique entre les dossiers et les fichiers sources

Au-delà de l'architecture, l'application web implémente une logique entre les fichiers sources qui permet de rendre le code source modulaire (*cf. Annexe Figure 2*). Si *app.R*

lance l'application sur un serveur local, ce lancement nécessite la création d'une variable UI et d'une variable Server. Or ces variables sont créées respectivement dans les dossiers *Server* et *UI* qui se trouvent dans le dossier *R*.

En effet, la variable UI permet de définir l'aspect interface de l'application. Il s'agit donc d'appeler des variables R qui définissent les DOMs (Document Object Model) de l'interface. Ces variables R sont soit conçues via une fonction qui enveloppe les balises HTML via le préfixe *tags\$* comme *tags\$p()* qui définit un DOM paragraphe selon les conventions HTML, soit des fonctions R Shiny tel que :

Action button/link

```
actionButton(inputId, label, icon = NULL, width = NULL, ...)  
actionLink(inputId, label, icon = NULL, ...)
```

Arguments

- inputId** The `input` slot that will be used to access the value.
- label** The contents of the button or link--usually a text label, but you could also use any other HTML, like an image.
- icon** An optional `icon()` to appear on the button.
- width** The width of the input, e.g. `'400px'`, or `'100%'`; see `validateCssUnit()`.
- ...** Named attributes to be applied to the button or link.

Description de la fonction `actionButton` de la bibliothèque R Shiny

Checkbox Input Control

```
checkboxInput(inputId, label, value = FALSE, width = NULL)
```

Arguments

- inputId** The `input` slot that will be used to access the value.
- label** Display label for the control, or `NULL` for no label.
- value** Initial value (`TRUE` or `FALSE`).
- width** The width of the input, e.g. `'400px'`, or `'100%'`; see `validateCssUnit()`.

Description de la fonction `checkboxInput` de la bibliothèque R Shiny

Toutefois, la variable UI n'est pas uniquement composé de variables DOM R mais aussi de variables script, qui inclut un script CSS ou un script JavaScript stocké dans le dossier *www* via les deux fonctions R suivantes :

Include Content From a File

```
includeHTML(path)  
  
includeText(path)  
  
includeMarkdown(path)  
  
includeCSS(path, ...)  
  
includeScript(path, ...)
```

Arguments

path The path of the file to be included. It is highly recommended to use a relative path (the base path being the Shiny application directory), not an absolute path.
... Any additional attributes to be applied to the generated tag.

Description des fonctions d'inclusions de fichiers externes

La variable UI qui est donc appelée par la fonction `shinyApp(ui = ui, server = server)` dans `app.R` est défini par le morceau de code suivant dans `createUI.R` dans le dossier `R/UI` :

```
# UI qui sera donné initialement à l'utilisateur  
ui <- fluidPage(  
  includeScript('www/JavaScript/adminModeProcess.js'),  
  
  includeScript('www/JavaScript/feuxProcess.js'),  
  includeScript('www/JavaScript/volesButtonsProcess.js'),  
  
  includeScript('www/JavaScript/setYearProcess.js'),  
  includeScript('www/JavaScript/endLoadingYearProcess.js'),  
  
  includeScript('www/JavaScript setTypeIndicatorProcess.js'),  
  includeScript('www/JavaScript setIndicatorProcess.js'),  
  includeScript('www/JavaScript setDeclinaisonProcess.js'),  
  includeScript('www/JavaScript setGroupeProcess.js'),  
  
  includeScript('www/JavaScript selectNotChangedProcess.js'),  
  
  includeScript('www/JavaScript selectTypeIndicatorProcess.js'),  
  includeScript('www/JavaScript selectIndicatorProcess.js'),  
  includeScript('www/JavaScript selectDeclinaisonProcess.js'),  
  includeScript('www/JavaScript selectGroupeProcess.js'),  
  
  includeScript('www/JavaScript accueilClickedTabProcess.js'),  
  includeScript('www/JavaScript accueilModeNewProcess.js'),  
  includeScript('www/JavaScript accueilModeOldProcess.js'),  
  includeScript('www/JavaScript indicatorNameUpdateProcess.js'),  
  includeScript('www/JavaScript idIndicatorAccueilFromNameProcess.js'),  
  
  includeScript('www/JavaScript idDataVizAccueilFromHoverProcess.js'),  
  
  includeCSS("www/CSS/enTeteCSS.css"),  
  includeCSS("www/CSS/settingsCSS.css"),  
  includeCSS("www/CSS/leftBandeauGlobalExpertCSS.css"),  
  includeCSS("www/CSS/generalCSS.css"),  
  includeCSS("www/CSS/leftBandeauAccueilCSS.css"),  
  includeCSS("www/CSS/leftBandeauAdminCSS.css"),  
  divTotal  
)
```

Avec la variable `divTotal` une variable R qui rassemble tous les variables DOM R généraux de l'interface.

Par ailleurs, la variable Server permet de définir les fonctionnalités de l'application web tel que l'affichage de visualisations de données, la navigation entre les divers onglets, le

dynamisme basique de certaines composantes de l'interface et la programmation des DOM input comme les boutons, les checkbox, les boutons radio, le slider etc. Pour cela, la variable Server est composée de reactiveValues, c'est-à-dire des variables objet à plusieurs champs qui peuvent être modifiées durant l'utilisation de l'application web et qui ne sont accessibles que dans des fonctions particulières (ex: observeEvent(), reactive(), observe() etc.), et des appels de fonctions R qui mettent en place les fonctionnalités et qui sont elles-mêmes créés dans des scripts R stockés dans le dossier *R/Server*.

Ainsi, la variable Server, qui est donc appelée avec la variable UI dans la fonction `shinyApp(ui = ui, server = server)` dans *app.R*, est créé dans *createServer.R* dans le dossier *R/Server*.

De plus, afin de pouvoir récupérer les données de la base de données sur laquelle l'application web est basée, le dossier *Datas* recensent des scripts R qui permettent la connexion avec la base de données et la création de fonctions réalisant la bonne requête SQL par rapport aux informations d'entrées sélectionnées par l'utilisateur via l'interface puis la réalisation de sa visualisation de données.

Or, pour pouvoir utiliser les fonctions et les variables conçues en dehors des scripts où ils sont créés, la logique réside dans celle d'une application R Shiny. En effet, comme l'explique le logigramme du lancement de l'application (cf. Annexes *Figure 2*), lors de

l'enclenchement du bouton  `Run App` dans l'environnement de développement RStudio, le fichier *app.R* va charger les scripts disponibles directement au premier niveau dans le dossier *R*. Ainsi, avant d'appeler des variables UI et Server qui n'ont pas été créés dans *app.R* et qui sont pour l'instant non existants, *app.R* charge le script *loadPackages.R* qui installe les bibliothèques nécessaires à l'application puis le script *loadScripts.R*. Ce script en particulier charge dans un ordre précis les scripts R présents dans les dossiers internes au dossier *R* et donc crée toutes les variables et fonctions nécessaires à l'application. Ce n'est qu'après avoir charger ces scripts qu'*app.R* lance son propre code et donc l'application.

C. Visualisation de l'application

L'interface de l'application se décompose en 3 pages principales : Accueil, Global et Expert.

La page Accueil rassemble les informations nécessaires à la compréhension de l'application et de ses indicateurs.

La page Global propose un ensemble d'indicateurs simples et un ensemble de choix de représentations limité destiné au grand public.

La page Experte reprend les mêmes principes que la page Global avec un accès élargi d'indicateurs et un ensemble de choix de représentations bien plus vaste et précis.



Indicateurs de la biodiversité AURA

[ACCUEIL](#)
GLOBAL
[EXPERT](#)

- [Renseignements](#)
- [Mode de représentation](#)
- [Indicateur de connaissance](#)
- [Indicateurs d'état de la biodiversité](#)
- [Indicateurs de pressions anthropiques](#)

Nos collaborateurs



PRÉFET
DE LA RÉGION
AUVERGNE-RHÔNE-ALPES
Léonard
Dessaint
Président



La Région
Auvergne-Rhône-Alpes



OFB
OFFICE FRANÇAIS
DE LA BIODIVERSITÉ



cbn
CONSERVATOIRE
BOTANIQUE NATIONAL
ALPIN



cbn
CONSERVATOIRE
BOTANIQUE NATIONAL
MASSIF CENTRAL



Flavia
Association pour les
Papillons et leur Étude



LPO
AGIR pour la
BIODIVERSITÉ
AUVERGNE-RHÔNE-ALPES



Page Accueil de l'application web



Indicateurs de la biodiversité AURA

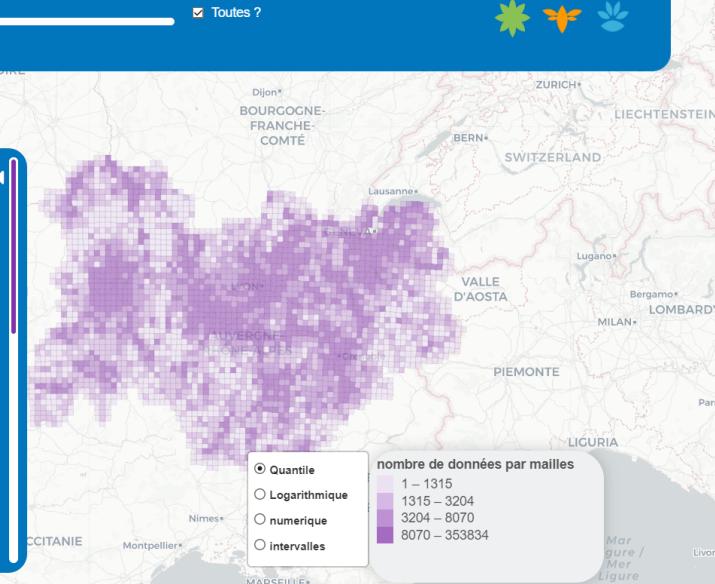
[ACCUEIL](#)
GLOBAL
[EXPERT](#)

Année : Toutes ?

Nombre de données : **1 571 365**

Nombre d'espèces : **25 481**

Indicateur :



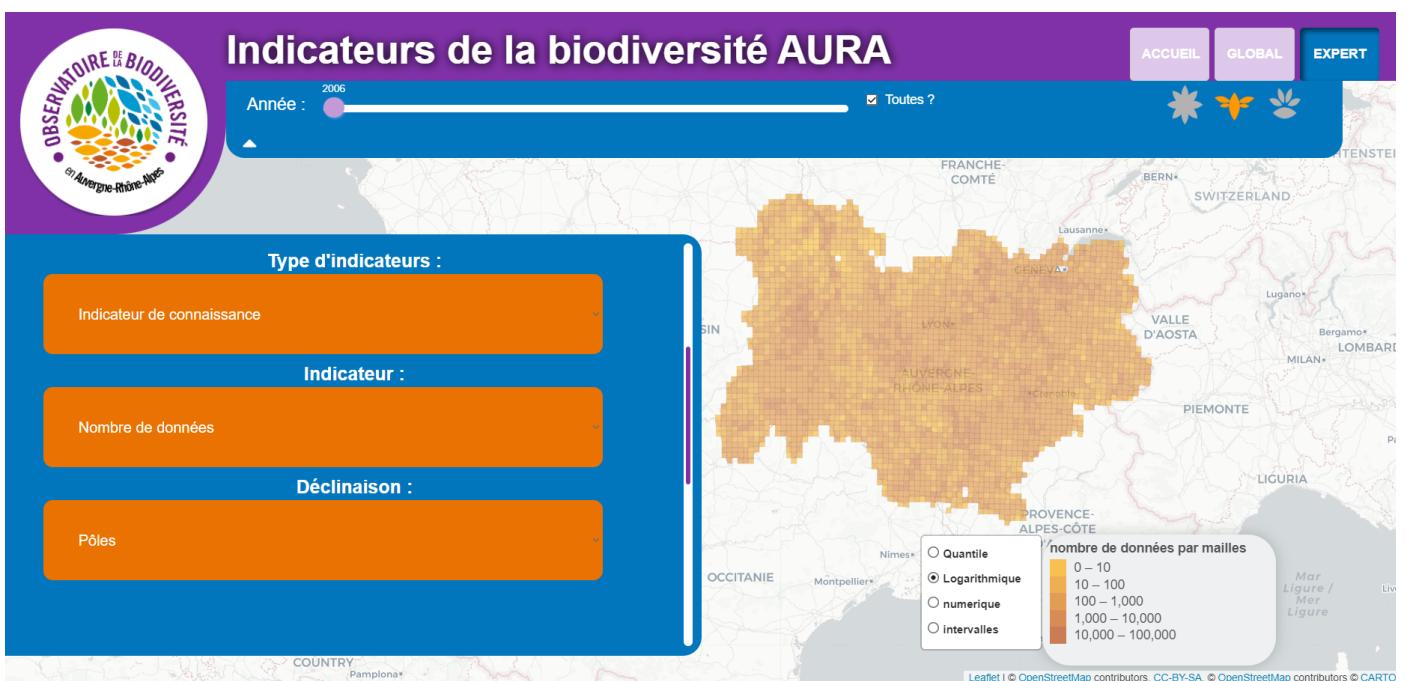
nombre de données par mailles

Quantile
Logarithmic
Numerical
Intervals

1 – 1315
1315 – 3204
3204 – 8070
8070 – 353834

Page Global de l'application web

7 sur 34

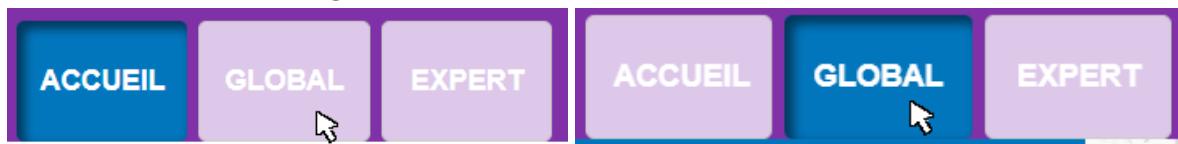


Page Expert de l'application web

D. Récapitulatif des interactions utilisateurs

a. Page d'accueil & navigation

- **Boutons de navigation**



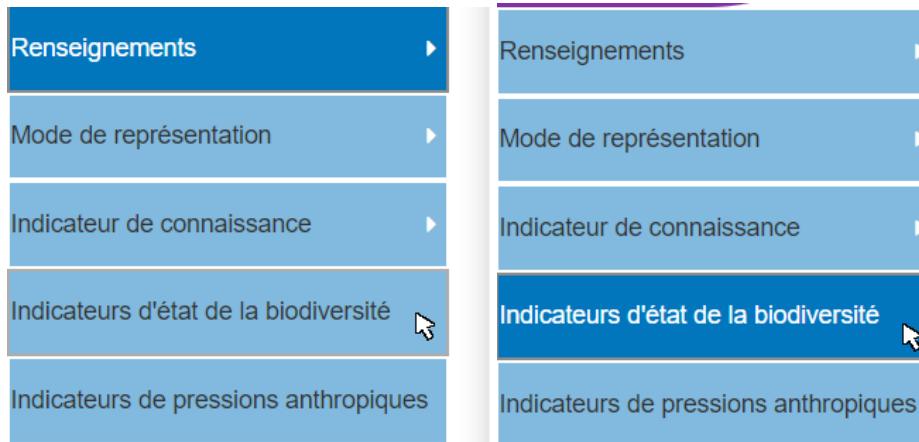
Fichiers concernés :

- Dossier R/Server :
 - *buttonsNavTabFct.R* : gère la programmation des boutons des onglets de chaque page

Variables importantes :

- *reactiveValues* : *data_page\$tpage* et *data_page\$fromPage*

- **Clic sur un indicateur ou une information**



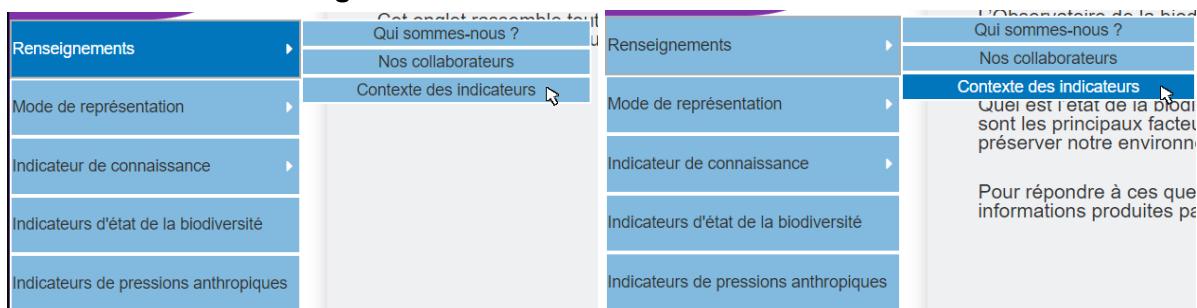
Fichiers concernés :

- Dossier *R/Server* :
 - *buttonsAccueilTabFct.R* : gère la programmation du bouton de la page accueil sélectionné par l'utilisateur
- Dossier *www/JavaScript* :
 - *accueilClickedTabProcess.js* : envoie à R l'id du bouton qui a été sélectionné par l'utilisateur
 - *accueilModeNewProcess.js* : change l'effet de style du bouton sélectionné
 - *accueilModeOldProcess.js* : change l'effet de style de l'ancien bouton sélectionné

Variables importantes :

- *reactiveValues* : *data_tabAccueil\$tab*
- input modifié par Javascript : *input\$tabAccueil>New*

● Clic sur une catégorie d'indicateur ou d'informations



Fichiers concernés :

- Dossier *R/Server* :
 - *buttonsAccueilTabFct.R* : gère la programmation du bouton de la page accueil sélectionné par l'utilisateur
- Dossier *www/JavaScript* :
 - *accueilClickedTabProcess.js* : envoie à R l'id du bouton qui a été sélectionné par l'utilisateur

- *accueilModeNewProcess.js* : change l'effet de style du bouton sélectionné
- *accueilModeOldProcess.js* : change l'effet de style de l'ancien bouton sélectionné

Variables importantes :

- *reactiveValues* : *data_tabAccueil\$tab*
- *input* modifié par Javascript : *input\$tabAccueil>New*

• Boutons de lien vers les collaborateurs



Fichiers concernés :

- Dossier *R/UI/body* :
 - *createLeftBandeauAccueil.R* : gère la programmation des DOM HTML nécessaires au corps de la page accueil. Il s'agit ici d'un *tags\$a(href="www.lien.com",imageDuLogo)* dans la construction de la grille de logo du panneau information lié au bouton "Nos Collaborateurs".

b. Page globale et experte

• Barre de choix de l'année



Fichiers concernés :

- Dossier *R/Server*
 - *yearsSettingsFct.R* : Déetecte une modification de la scrollBar. Elle permet aussi d'initialiser la scrollBar à un état et de demander l'actualisation de l'état de *data_year\$loading*. Modifie également les données à afficher si nécessaire.
- Dossier *www/JavaScript*
 - *setYearProcess.js* : Demande l'actualisation de la valeur des input en fonction d'une année demandée.
 - *endLoadingYearProcess.js* : Actualise la valeur de *data_year\$loading*.

Variables importantes :

- *input* : *input\$yearScrollbar*

- input modifié par javascript : *input\$setYear*
- input modifié par javascript : *input\$endLoadingYear*
- reactiveValues : *data_year\$year*
- reactiveValues : *data_year\$loading* : permet de ne pas actualiser les données si ce n'est pas nécessaire.

- Boîte de sélection de toutes les années



Fichiers concernés :

- Dossier R/Server
 - *yearsSettingsFct.R* : Déetecte une modification de la check box. Elle permet aussi d'initialiser la check box à un état et de demander l'actualisation de l'état de *data_year\$loading*. Modifie également les données à afficher si nécessaire.
- Dossier www/JavaScript
 - *setYearProcess.js* : Demande l'actualisation de la valeur des input en fonction d'une année demandée.
 - *endLoadingYearProcess.js* : Actualise la valeur de *data_year\$loading*.

Variables importantes :

- input : *input\$checkAllYears*
- input modifié par javascript : *input\$setYear*
- input modifié par javascript : *input\$endLoadingYear*
- reactiveValues : *data_year\$year*
- reactiveValues : *data_year\$loading* : permet de ne pas actualiser les données si ce n'est pas nécessaire.

- Boutons de pôles



Fichiers concernés :

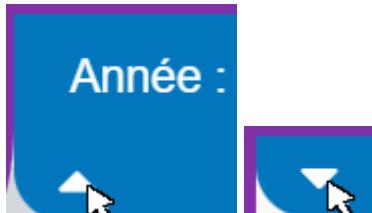
- Dossier R/Server
 - *buttonsPolesSettingsFct.R* : Déetecte un clic sur l'un des boutons et agit en conséquence pour actualiser leur valeur et leur couleur. Demande aussi l'actualisation des feux de pôles et la modification des sélecteurs de déclinaison et de groupe.
- Dossier www/JavaScript
 - *polesButtonsProcess.js* : Actualise la couleurs des sélecteurs en fonction du/des pôles sélectionnés.

Variables importantes :

- inputs : *input\$actionButtonFlower*, *input\$actionButtonBee* et *input\$actionButtonPaw*

- reactiveValues : *data_polesButtons\$flore*, *data_polesButtons\$invertebre*, *data_polesButtons\$vertebre* : mémorise le/les pôles actifs.

- Bouton de réduction du bandeau de settings



Fichiers concernés :

- Dossier R/Server
 - *buttonsCollapseFct.R* : Déetecte un clic sur le bouton de collapse ou de uncollapse.

Variables importantes :

- inputs : *input\$collapseSettingsButton* et *input\$uncollapseSettingsButton*

- Bouton de réduction du bandeau des graphiques



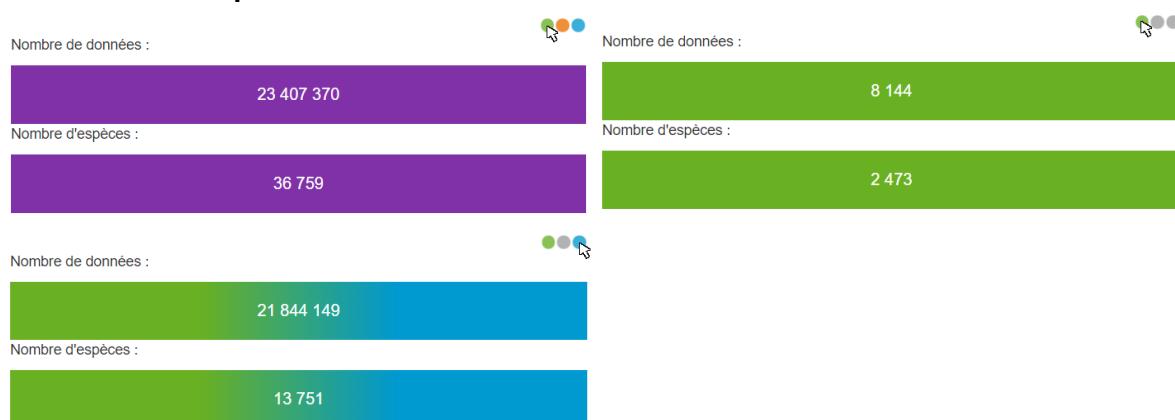
Fichiers concernés :

- Dossier R/Server
 - *buttonsCollapseFct.R* : Déetecte un clic sur le bouton de collapse ou de uncollapse. Lors du uncollapse, ré-actualise également la couleurs et la valeurs des éléments affichés qui ont été mémorisés.

Variables importantes :

- inputs : *input\$collapseBandeauButton* et *input\$uncollapseBandeauButton*
- reactiveValues : *data_page\$page* et *data_page\$fromPage* : permet de savoir s'il faut afficher le bandeau global ou expert.

- Feux des pôles



Fichiers concernés :

- Dossier R/Server
 - *buttonsPolesLeftBandeauFct.R* : Déetecte un clic sur l'un des feux et agit en conséquence pour actualiser leur valeur et leur couleur.
Demande aussi l'actualisation des couleurs de l'affichage des nombres et leur valeur.
- Dossier www/JavaScript
 - *feuxProcess.js* : actualise la couleurs des l'affichage des nombres et demande pour l'actualisation de leur valeur.
 - *feuxNumbersProcess.js* : demande l'actualisation de la valeur des nombres.

Variables importantes :

- inputs : *input\$feuFlore*, *input\$feuInvertebre* et *input\$feuVertebre*
- reactiveValues : *data_polesFeux\$flore*, *data_polesFeux\$invertebre*, *data_polesFeux\$vertebre* : mémorise le/les pôles actifs.
- reactiveValues : *data_polesButtons\$flore*, *data_polesButtons\$invertebre*, *data_polesButtons\$vertebre* : détermine les feux activables.

• Clic sur un graphique et pop up des graphiques



Fichiers concernés :

- Dossier R/Server :
 - *popUpDataVizFct.R* : gère la programmation de l'apparition de la pop up en fonction d'un input modifié par JavaScript
- Dossier www/JavaScript :
 - *idDataVizAccueilOnClickProcess.js* : écoute les clics de l'utilisateur et renvoie l'id de la visualisation de donnée cliquée

Variables importantes :

- input modifié par Javascript : *input\$idDataViz*
- Variable globale <graph>Copy : graphique à afficher dans le popup

c. Ensemble des sélecteurs

• Sélecteurs de type d'indicateur

Fichiers concernés :

- Dossier *R/Server* :
 - *modifySelectors.R* : gère le remplissage et le choix d'un type d'indicateur.
 - *selectIndicators.R* : gère les interactions utilisateur avec le sélecteur.
- Dossier *R/UI/body* :
 - *createListIndicators.R* : crée la liste des types d'indicateurs qui devront être remplis dans le sélecteur.
- Dossier *www/JavaScript* :
 - *selectTypeIndicatorProcess.js* : permet d'afficher un type d'indicateur sélectionné par le programme.
 - *setTypeIndicatorProcess.js* : permet de sélectionner un type d'indicateur, en fonction de l'indicateur pour un changement depuis la page globale.
 - *selectNotChangedProcess.js* : permet de continuer la modification des sélecteurs même si ce sélecteur n'a pas été modifié.

Variables importantes :

- input : *input\$selectTypeIndicator*
- input modifié par javascript : *input\$selectNotChanged*
- reactiveValues : *data_currentInd\$typeInd* : mémorisent le type d'indicateur courant.
- Variable globale list *listTypesIndicators*: ensemble des types d'indicateurs disponibles. Permet de remplir le sélecteur correctement.
- reactiveValues : *data_page\$page* et *data_page\$fromPage* : permettent de savoir si le sélecteur est présent sur le dashboard

● Sélecteur d'indicateur (seul présent en page globale)



Fichiers concernés :

- Dossier *R/Server* :
 - *modifySelectors.R* : gère le remplissage et le choix d'un indicateur en fonction du type d'indicateur.
 - *selectIndicators.R* : gère les interactions utilisateur avec le sélecteur.
- Dossier *R/UI/body* :
 - *createListIndicators.R* : crée la liste des indicateurs (et leur dépendance avec les pôles, types, etc.) qui devront être remplies dans le sélecteur.
- Dossier *R/Data* :

- *datasForServerFct.R* : crée les graphiques en fonction de l'indicateur choisi.
- Dossier *www/JavaScript* :
 - *indicatorNameUpdateProcess.js* : modifie le nom de l'indicateur affiché sous le/les sélecteurs.
 - *selectIndicatorProcess.js* : permet d'afficher un indicateur sélectionné par le programme.
 - *setIndicatorProcess.js* : permet de sélectionner un indicateur, en fonction du type d'indicateur.
 - *selectNotChangedProcess.js* : permet de continuer la modification des sélecteurs même si ce sélecteur n'a pas été modifié.

Variables importantes :

- *input* : *input\$selectIndicator*
- *input* modifié par javascript : *input\$selectNotChanged*
- *reactiveValues* : *data_currentInd\$indicator* et *data_currentInd\$indicatorName* : mémorisent l'indicateur courant et son nom.
- *reactiveValues* : *data_currentIndicator\$hasChanged* : permet de mémoriser si l'indicateur a été modifié et si les données doivent être calculées à nouveau.
- Variable globale *data.frame tabIndicators* : ensemble des indicateurs disponibles et leurs dépendances aux types, déclinaison, etc. Permet de remplir le sélecteur correctement.
- *reactiveValues* : *data_page\$page* et *data_page\$fromPage* : permettent de savoir si le sélecteur est présent sur le dashboard

- **Sélecteurs de déclinaison**

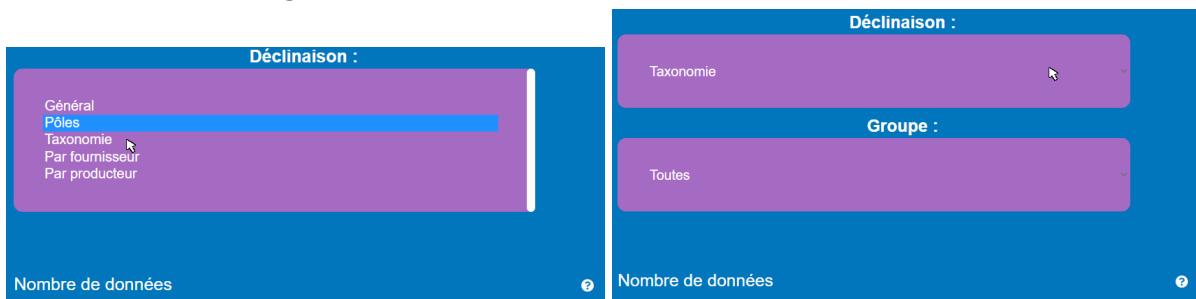
Fichiers concernés :

- Dossier *R/Server* :
 - *modifySelectors.R* : gère le remplissage et le choix d'une déclinaison.
 - *selectIndicators.R* : gère les interactions utilisateur avec le sélecteur.
- Dossier *R/UI/body* :
 - *createListIndicators.R* : crée la liste des déclinaisons qui devront être remplies dans le sélecteur.
- Dossier *R/Data* :
 - *datasForServerFct.R* : crée les graphiques en fonction de la déclinaison choisie.
 -
- Dossier *www/JavaScript* :
 - *selectDeclinaisonProcess.js* : permet d'afficher une déclinaison sélectionnée par le programme.
 - *setDeclinaisonProcess.js* : permet de sélectionner une déclinaison, en fonction de l'indicateur courant.
 - *selectNotChangedProcess.js* : permet de continuer la modification des sélecteurs même si ce sélecteur n'a pas été modifié.

Variables importantes :

- input : *input\$selectDeclinaison*
- input modifié par javascript : *input\$selectNotChanged*
- reactiveValues : *data_currentInd\$declinaison* : mémorisent la déclinaison courante.
- reactiveValues : *data_currentIndicator\$hasChanged* : permet de mémoriser si la déclinaison a été modifiée et si les données doivent être calculées à nouveau.
- Variable globale data.frame *declinaisonIndicator* : ensemble des déclinaisons disponibles et leur nom. Permet de remplir le sélecteur correctement.
- reactiveValues : *data_page\$page* et *data_page\$fromPage* : permettent de savoir si le sélecteur est présent sur le dashboard

- **Sélecteur de groupe (ne s'active que pour la déclinaison taxonomie)**



Fichiers concernés :

- Dossier *R/Server* :
 - *modifySelectors.R* : gère le remplissage et le choix d'un groupe.
 - *selectIndicators.R* : gère les interactions utilisateur avec le sélecteur.
- Dossier *R/UI/body* :
 - *createListIndicators.R* : crée la liste des groupes (ainsi que leur liens avec les pôles) qui devront être remplis dans le sélecteur.
- Dossier *R/Datas* :
 - *datasForServerFct.R* : crée les graphiques en fonction du groupe choisi.
 -
- Dossier *www/JavaScript* :
 - *selectGroupeProcess.js* : permet d'afficher un groupe sélectionné par le programme.
 - *setGroupeProcess.js* : permet de sélectionner un groupe, en fonction de la déclinaison courante.
 - *selectNotChangedProcess.js* : permet de continuer la modification des sélecteurs même si ce sélecteur n'a pas été modifié, et d'appeler la fonction *datasForServerFct* ou *dispDatasForServerFct* (dans le fichier *datasForServerFct*).

Variables importantes :

- input : *input\$selectGroupe*
- input modifié par javascript : *input\$selectNotChanged*
- reactiveValues : *data_currentInd\$groupe* : mémorisent le groupe courant.

- Variable globale `data.frame tabGroupe`: ensemble des groupes disponibles ainsi que leurs liens avec les pôles. Permet de remplir le sélecteur correctement.
- `reactiveValues` : `data_page$page` et `data_page$fromPage` : permettent de savoir si le sélecteur est présent sur le dashboard

E. Récapitulatif des variables de stockage de données

- **ReactiveValues (*R/Server/createServer.R*) :**
 - `data_page { page = "accueil", fromPage = "accueil" }` : connaissance de la page précédente et de la page actuelle
 - `data_year { year = 0, loading = FALSE }` : mémorisation de l'année à observer
 - `data_polesButtons { flore = TRUE, invertébre = TRUE, vertébre = TRUE }` : connaissance de l'état d'activation de chaque bouton de pôles
 - `data_polesFeux { flore = TRUE, invertébre = TRUE, vertébre = TRUE }` : connaissance de l'état d'activation de chaque bouton de feux
 - `data_currentInd { typeInd = listTypesIndicators[[1]], indicator = tabIndicators[1,2], indicatorName = tabIndicators[1,1], declinaison = declinaisonIndicator[1,2], groupe = tabGroupe[1,1], hasChanged = FALSE }` : connaissance de l'indicateur actuel
 - `data_tabAdmin { tab = "RAJOUT" }` : connaissance de l'onglet actuel dans la page Administrateur si celle-ci est activée (**UTILE POUR LA PAGE ADMIN**)
 - `data_tabAccueil { tab = "ind13" }` : connaissance de l'onglet actuel dans la page Accueil
 - `data_nbInd { nbGeneral = 5, nbSub = list(3,4,3,0,0) }` : connaissance du nombre global d'indicateurs et du nombre de sous-indicateur pour un indicateur général (**UTILE POUR LA PAGE ADMIN**)
- **Variables globales pour les types d'indicateurs, indicateurs, déclinaisons et groupes (*R/UI/body/createListIndicators.R*) :**
 - `list listTypesIndicators` : liste recensant tous les types d'indicateurs.
 - `data.frame tabIndicators(indName, ind, isGlobal, isExpert, typeInd, isPoles, isTaxo, isFournisseur, isProducteur, isAnnees)` :
 - `indName` est le nom affiché de l'indicateur et `ind` sa valeur (string simplifié) associée.
 - `isGlobal` permet de définir qui doivent être présent dans la page globale.
 - `isExpert` définit les indicateurs “spéciaux”. Ici, se sont les indicateurs qui doivent être présents pour la représentation par taxonomie dans la

page globale. Par exemple, “donnéesTaxo” en réalité “données” avec la déclinaison “taxo” dans la page experte.

- typeInd est le type d'indicateur dont fait partie l'indicateur en question.
- isPoles, isTaxo, isFournisseur et isProducteur définissent les déclinaisons possibles de l'indicateur. La déclinaison “générale” est toujours possible.
- isAnnees n'est pas utilisé dans cette version de l'application mais permettrait de discriminer des indicateurs en fonction de s'ils peuvent être représentés en fonction de toutes les années. En pratique, on utilise data_currentInd\$indicator et seulement “especes” ne peut pas être représenté de la sorte.
- data.frame declinaisonIndicator(name, value) : liste de tous les noms de déclinaisons possibles (général, pôles, taxonomie, fournisseur, producteur) et une valeur (string plus simple) associée.
- data.frame tabGroupe(groupe, isFlore, isInvertebre, isVertebre) :
 - groupe est une taxonomie parmi toutes celles disponibles en plus de “rien” (liée à aucun pôle) et “Toutes” (liée à tous les pôles).
 - isFlore, isInvertebre, isVertebre définissent le pôle lié à une taxonomie. Par exemple, on pourra avoir (“Toutes”, TRUE, TRUE, TRUE) ou (“Oiseaux”, FALSE, FALSE, TRUE).

- **Variables globales pour les graphiques (*R/Datas/datasForServerFct.R*) :**

- leaflet() mapPlot : carte qui se trouve dans le fond, elle est toujours affichée si on est dans la page globale ou expert mais peut ne contenir que le fond de carte.
- list(nbPlot, plot1, plot2, titre1, titre2) piePlot : liste de 2 pie chart. Les deux pie charts ne servaient que dans le cas de deux pôles activés simultanément. nbPlot définit le nombre de pie chart disponibles.
- list(isPlot, plot, titre) histoPlot : liste avec 1 histogramme qui est tracé si isPlot est vrai (si l'histogramme est disponible).
- list(isPlot, plot, titre) barPlot : liste avec 1 bar chart qui est tracé si isPlot est vrai (si le bar chart est disponible).
- list(nbPlot, plot1, plot2, titre1, titre2) dataFournProd : liste de 2 bar chart. Les deux bar charts correspondent au nombre de données et au nombre de jeux de données. nbPlot définit le nombre de bar chart disponibles (0 ou 2 actuellement).

II. La maintenance des fichiers sources

A. Maintenir une information dans la page accueil

Pour maintenir une information dans la page accueil, il faut déjà savoir comment l'accueil s'organise puis déterminer la manière dont vous souhaitez mettre en avant cette information selon cette organisation prédéfinie.

En effet, afin de proposer un accueil clair et facile à prendre en main, la navigation est en réalité une possible double navigation : une option de la navigation principale peut faire apparaître une nouvelle navigation subsidiaire à l'option choisie si cette option possède une icône en forme de flèche pointant vers la droite.

Cette forme particulière de navigation impose alors un système à deux niveaux de précisions. Pour mentionner les **options du premier niveau**, nous parlerons **d'informations générales**, et pour **ceux du second niveau**, de **catégories d'informations**. En effet, les informations générales rassemblent des concepts généraux tandis que les catégories d'informations énoncent des explications sur un ensemble de sous-informations.

Ainsi, avant d'ajouter, supprimer ou modifier une information dans la page accueil, il faut connaître le statut de cette information dans ce système à deux niveaux. Toutefois, si vous comprenez l'ajout d'une information, la suppression et la modification en découlent. Nous ne traiterons donc seulement les deux cas suivants : l'ajout d'une information générale sans catégorie d'informations et l'ajout d'une catégorie d'information.

a. Ajouter une information générale sans catégorie d'informations

Les étapes 1 à 3 doivent se réaliser à l'intérieur du script *createLeftBandeau.R* en respectant l'ordre d'apparition des variables afin de conserver un code clair et compréhensible pour le suivant. En effet, il faut éviter de créer un panneau d'informations parmi la création de boutons. Conservons au plus l'organisation du script.

1) Créer le bouton de sélection de cette information

Pour pouvoir permettre à l'utilisateur de cliquer sur l'information via la barre de navigation la première étape est de créer un bouton. Toutefois, ce travail ne doit pas être réalisé n'importe comment. La création de ce bouton doit suivre une convention pour que tous les scripts puissent se référer correctement à lui.

Voici un modèle de bouton que l'on peut trouver dans l'application web :

```
# Sélecteur d'indicateur unique
buttonInd5<-actionButton(inputId="ind5",
                           class="singleInd",
                           label="Indicateurs de pressions anthropiques")
```

Vous pouvez remarquer dans cet exemple, que le bouton doit être défini par la fonction *actionButton()* et non *tags\$button()*.

De plus, son id doit correspondre à un format tel que : "ind_" avec à la place du tiret du bas un numéro qui n'a pas été déjà pris par d'autres boutons. Le meilleur moyen est donc de se positionner à la fin de la zone de création de boutons et d'incrémenter le dernier id.

Par la suite, puisqu'il s'agit d'une information générale, votre bouton doit se référer comme appartenant à la classe "*singleInd*".

Par ailleurs, l'attribut du label correspond au titre de votre bouton et donc de votre information générale.

Enfin, le nom de votre variable doit aussi coïncider au maximum avec cet exemple afin de rendre le code clair et maintenable.

2) Ajouter ce bouton dans la barre de navigation

Une fois le bouton créé, il est nécessaire de l'assigner à la barre de navigation pour qu'il apparaisse dans l'interface. Puisqu'il s'agit d'une information générale, la démarche est simple.

Voici à quoi peut rassembler la variable stockant la barre de navigation de la page accueil :

```
# Navigation des sélecteurs de l'ensemble des sélecteurs des indicateurs
divNavBar <- div(id="navBar"
                  , divSubNav1
                  , divSubNav2
                  , divSubNav3
                  , buttonInd4
                  , buttonInd5)
```

Comme vous pouvez le remarquer, cette barre de navigation est unique et doit contenir tous les blocs qui définissent la sélection de vos informations. Il y en a précisément deux types. Pour le cas d'une information générale, seul le nom de la variable stockant votre bouton vous intéresse. En effet, il suffit de rajouter cette variable en bas de la liste pour que la nouvelle option apparaissent en bas de la barre de navigation.

3) Créer le panneau d'information associé au bouton

Une fois que le bouton est implémenté, il faut absolument programmer ce qu'il doit faire apparaître une fois cliqué. Il est donc temps de créer un panneau d'informations en suivant les conventions à la lettre pour s'assurer un bon fonctionnement.

Voici un modèle simple de panneau d'informations :

```
divInfo_ind5 <- div(id="info_ind5",
                      class="info",
                      h1(class="indName",
                          "Indicateurs de pressions anthropiques"),
                      tags$p(class="indExplanation",
                            "Lorem ipsum dolor sit amet,
                            consectetur adipiscing elit.
                            Quisque sodales nunc eu egestas
                            dictum. Mauris maximus ante et
                            diam lacinia sodales. Ut
                            elementum tortor sed lorem commodo,
                            ut aliquet mauris viverra. In vel
                            odio ut magna pretium imperdiet eu
                            et erat. Curabitur ornare cursus mi,
                            eget vulputate orci pellentesque vel."))
```

Vous pouvez donc observer que la variable qui stocke votre panneau d'information doit se conformer à cette convention de nommage suivante : "*divInfo_ind_*" avec le numéro qui correspond à l'id de votre bouton. De même pour son id qui doit respecter le format "*info_ind_*".

De plus, ce panneau doit appartenir à la classe "*info*" et doit comporter obligatoirement un DOM titre h1 créé par la fonction *h1()* dont la classe se réfère à "*indName*" et dont l'attribut du label est identique à celui de votre bouton.

Enfin, afin de pouvoir énoncer votre explication, créez à la suite autant de variables avec la fonction *tags\$p* et la classe "*indExplanation*". Si vous souhaitez sauter des lignes, il est nécessaire de couper votre explication en deux variables *tags\$p* et d'y intercaler la fonction *br()*.

4) Rajouter dans les listes du script *buttonsAccueilTabFct.R* l'id du bouton de sélection puis la variable stockant le panneau d'informations respectivement

Une fois toutes les variables nécessaires créées, la dernière étape consiste à permettre au script *buttonsAccueilTabFct.R* qui gère la navigation de la page accueil de pouvoir reconnaître le bon panneau d'informations à afficher en fonction de l'id du bouton qui a été cliqué. Pour cela, il suffit de rajouter à la fin des deux listes suivantes respectivement l'id du bouton qui doit suivre le format "*ind_*" puis le nom de la variable du panneau information associé qui doit suivre le format "*divInfo_ind_*" :

```
# Listes nécessaires à la bonne indexation des panneaux infos
listIdTabAccueil <- list("ind1", "ind11", "ind12", "ind13", "ind2", "ind21", "ind22", "ind23", "ind24", "ind3",
                         "ind31", "ind32", "ind33", "ind4", "ind5")
listDivTabAccueil <- list(divInfo_ind1, divInfo_ind11, divInfo_ind12, divInfo_ind13,
                           divInfo_ind2, divInfo_ind21, divInfo_ind22, divInfo_ind23, divInfo_ind24,
                           divInfo_ind3, divInfo_ind31, divInfo_ind32, divInfo_ind33,
                           divInfo_ind4, divInfo_ind5)
```

Une fois ce dernier détail réglé, votre nouvelle information devrait apparaître correctement dans la page accueil en fin de barre de navigation et devrait afficher le panneau d'informations remplies par vos soins à son clic.

b. Ajouter une catégorie d'informations

Pour ajouter une catégorie d'informations, cela revient à ajouter une information générale avec un peu plus de complexité. Revoyons rapidement les étapes ensemble.

1) Créer le bouton de sélection de cette information

Pour pouvoir permettre à l'utilisateur de cliquer sur la catégorie d'information via la barre de navigation, la première étape est de créer un bouton parent capable d'afficher le second niveau de la barre de navigation. Toutefois ce bouton sera différent d'un bouton d'informations générales classique. C'est pourquoi, si vous souhaitez transformer une information générale sans catégorie en une information générale avec au moins une catégorie, vous devez modifier votre bouton précédent pour rendre votre ancienne option unique en une option capable de se subdiviser.

Voici un modèle de bouton offrant la possibilité d'accéder au second niveau de la barre de navigation :

```
# Sélecteur d'indicateur unique
buttonInd1<-actionButton(inputId="ind1",
                           class="multipleInd",
                           div(class="labelMultipleInd",
                               id="ind1",
                               tags$p("Renseignements"),
                               icon(name = "caret-right")))
```

Vous pouvez remarquer que ce bouton partage beaucoup avec le bouton vu précédemment. Toutefois certains détails diffèrent. Parmi eux, le premier est évidemment sa classe qui n'est plus celle d'un "*singleInd*" mais d'un "*multipleInd*". En second lieu, l'attribut label n'est plus renseigné. A la place, une variable div est créée avec la fonction *div()*. Ce DOM div se réfère alors par la classe "*labelMultipleInd*" mais aussi par **son id qui est identique à celui du bouton**. A l'intérieur de ce DOM div, on y renseigne le titre de l'information via une fonction *tags\$p()* ainsi qu'un DOM i créé par la fonction *icon()* qui fournira l'icône de la flèche pointant vers la droite.

Toutefois, ce bouton ne suffit pas. En effet, il s'agit du bouton parent à votre catégorie d'informations. Si votre catégorie d'informations s'insère parmi d'autres catégories d'informations déjà existantes, ce bouton existe déjà. Si vous insérez la toute première catégorie d'information, il faut soit transformer le bouton parent qui était un "*singleInd*" ou le créer directement si cette catégorie est créée en même temps que l'information générale dont il est le parent.

Dans tous les cas, il faut procéder à la création d'un autre bouton : celui qui correspondra à la sélection de votre catégorie d'informations.

Voici un modèle de bouton qui se glisse dans le second niveau de la barre de navigation :

```
# Sélecteur de sous-indicateur
buttonInd11<-actionButton(inputId="ind11",
                            class="subInd",
                            label="Qui sommes-nous ?")
```

Ce bouton est très ressemblant aux deux autres mais ses différences sont primordiales. Cette fois-ci, son format de nom de variable est tel que : "*buttonInd__*" avec pour premier tiret du bas l'indice du bouton parent dont il découle et le second son propre indice. Ainsi la première catégorie d'informations de la première information générale prendra comme nom de variable celui de l'exemple. De même pour son id qui doit suivre le format "*ind__*". Par ailleurs, sa classe n'est ni celle d'un "*singleInd*" ni celle d'un "*multipleInd*" mais celle d'un "*subInd*". Enfin, son attribut label permet de renseigner le titre de cette catégorie d'informations.

2) Ajouter ce bouton dans la barre de navigation

Une fois le bouton créé, il est nécessaire de l'assigner à la barre de navigation pour qu'il apparaisse dans l'interface. Puisqu'il s'agit d'une catégorie d'informations, la démarche est plus complexe.

Rappelez-vous à quoi ressemble la barre de navigation dans l'application :

```
# Navigation des sélecteurs de l'ensemble des sélecteurs des indicateurs
divNavBar <- div(id="navBar"
  , divSubNav1
  , divSubNav2
  , divSubNav3
  , buttonInd4
  , buttonInd5)
```

Vous connaissez déjà le système pour une information générale qui ne posséderait aucune catégorie. Le second type de variable qui est acceptable est celle de la forme : "divSubNav_" avec pour tiret du bas l'indice du bouton parent. En effet, il s'agit d'une boîte qui englobe le bouton parent et les boutons des catégories ensemble.

Voici à quoi peut ressembler une variable divSubNav :

```
# Navigation des sélecteurs des sous-indicateurs de l'indicateur multiple
divSubNav1<-div(class="subNav",
  id="subNavInd1",
  buttonInd1,
  divSubNavContent1)
```

Comme expliqué précédemment, cette variable contient le bouton parent et une autre variable qui suit la forme : "divSubNavContent_" avec pour tiret du bas l'indice du bouton parent. Il s'agit du DOM div qui englobe l'ensemble des catégories d'informations qui descend du bouton parent qui correspond à l'indice.

Voici à quoi peut ressembler une variable divSubNavContent :

```
# Boîte rassemblant les sélecteurs des sous-indicateurs
divSubNavContent1<-div(class="subNavContent",
  id="subNavContentInd1",
  buttonInd11,
  buttonInd12,
  buttonInd13)
```

Ainsi, si vous souhaitez ajouter une catégorie d'informations à la suite de catégories d'informations déjà existantes, la seule modification utile pour faire apparaître le bouton de la catégorie dans le second niveau de la barre de navigation est de rajouter le bouton de la catégorie dans le SubNavContent associé à son parent.

Toutefois, s'il s'agit de la toute première catégorie d'informations, il est probable que ces structures n'existent pas et qu'il est nécessaire de les rajouter en précisant bien dans le divSubNavContent le bouton de la catégorie et dans le divSubNav le divSubNavContent associé ainsi que le bouton parent. Il suffira alors de passer cette variable divSubNav à la barre de navigation.

3) Créez le panneau d'information associé au bouton

Une fois que le bouton est implémenté, il faut absolument programmer ce qu'il doit faire apparaître une fois cliqué. Il s'agit de la même étape que pour l'ajout d'une information générale sans catégorie à une précision près : les conventions de nommage sont "*divInfo_ind_*" avec le double numéro qui correspond à l'id de votre bouton catégorie, et de même pour son id qui doit respecter le format "*info_ind_*".

4) Rajouter dans les listes du script *buttonsAccueilTabFct.R* l'id du bouton de sélection puis la variable stockant le panneau d'informations respectivement

Une fois toutes les variables nécessaires créées, la dernière étape consiste à permettre au script *buttonsAccueilTabFct.R* qui gère la navigation de la page accueil de pouvoir reconnaître le bon panneau d'informations à afficher en fonction de l'id du bouton qui a été cliqué. Pour cela, il suffit de rajouter à la fin des deux listes suivantes respectivement l'id du bouton qui doit suivre le format "*ind_*" puis le nom de la variable du panneau information associé qui doit suivre le format "*divInfo_ind_*" :

B. Maintenir un type d'indicateur

a. Ajouter un type d'indicateur dans les sélecteurs

Afin d'ajouter un type d'indicateur dans le sélecteur associé, il faut tout d'abord comprendre la logique de l'implémentation de ces derniers dans l'application. Pour cela, référez-vous à la partie du récapitulatif des variables de stockage de données.

Ensuite, rendez-vous dans le fichier *R/UI/body/createListIndicators.R* et cherchez la balise suivante :

-!!! DERNIERE LIGNE TYPE INDICATEUR !!!-

Au niveau de cette balise, ajoutez la ligne selon le modèle suivant (défini dans la structure du stockage de données) :

, "Indicateurs de pressions anthropiques"

Votre type d'indicateur est maintenant présent dans votre sélecteur dans la page expert.

b. Ajouter un type d'indicateur dans la page d'accueil

Afin d'ajouter un type d'indicateur dans la page d'accueil, se référer à "ajouter une information générale sans catégorie d'informations" et effectuer les mêmes étapes en respectant les informations remplies pour ajouter le type d'indicateur dans les sélecteurs.

C. Maintenir un indicateur

a. Ajouter un indicateur dans les sélecteurs

Afin d'ajouter un indicateur dans le sélecteur associé, il faut tout d'abord comprendre la logique de l'implémentation de ces derniers dans l'application. Pour cela, référez vous à la partie du récapitulatif des variables de stockage de données.

Ensuite, rendez-vous dans le fichier *R/UI/body/createListIndicators.R* et cherchez la balise suivante :

```
# ---!!! DERNIERE LIGNE INDICATEUR !!!!--- #
```

Au niveau de cette balise, ajoutez deux lignes selon le modèle suivant (défini dans la structure du stockage de données) :

```
# indName, ind, isGlobal, typeInd, isPoles, isTaxo, isFournisseur, isProducteur, isAnnees  
newRow <- list("Nombre d'espèces", "especes", TRUE, TRUE, "Indicateur de connaissance", TRUE, TRUE, FALSE, FALSE, TRUE);  
tabIndicators <- rbind(tabIndicators, newRow);
```

Votre indicateur est maintenant présent dans votre sélecteur selon les contraintes que vous lui avez définies.

b. Ajouter une représentation d'un indicateur

Pour ajouter une représentation, parmi celles déjà disponibles, d'un indicateur, il faut se plonger dans le code. En effet, il faut se rendre dans le script et la fonction associée à représentation que vous souhaitez.

Par exemple si vous voulez ajouter un histogramme du nombre de données en fonction des années pour l'indicateur "nombreEspeciesEnvahissantes", il vous faut vous intéresser à la fonction `afficher_hist` et à tous les tests if liés au type (qui représente la valeur de l'indicateur) pour rajouter un cas pour `type == "nombreEspeciesEnvahissantes"`. Par exemple, dans ce cas, on pourrait imaginer les lignes suivantes :

```
56+ else if (type == "nombreEspeciesEnvahissantes") {  
57   commande = paste("SELECT nb_espece_envahiss FROM", base)  
58   subTitre <- "du nombre d'espèces envahissantes";  
59 }
```

Néanmoins, cela peut ne pas être suffisant. Par exemple si cet indicateur était sur la base "orb_indicateurs.mv_sraddet_ind_enva", il faudrait également changer des lignes au niveau du remplissage de la base de données comme cela :

```

35 if ((groupe == "general") || (groupe=="pole")){
36   if (type == "nombreEspecesEnvahissantes") {
37     base = "orb_indicateurs.mv_sraddet_ind_enva"
38   }
39   else {
40     base = "orb_indicateurs.mv_sraddet_ind_pole"
41   }
42 }

```

En conclusion, pour prendre en compte correctement une nouvelle donnée, par exemple un indicateur, dans un des différents affichages de graphiques, il faut modifier le code de façon à prendre en compte ses dépendances aux différents pôles, déclinaisons, groupes et aux années mais aussi à la base de données qui lui est associée.

c. Ajouter un indicateur dans la page d'accueil

Afin d'ajouter un indicateur dans la page d'accueil, se référer à “ajouter une catégorie d'information” et effectuer les mêmes étapes en respectant les informations remplies pour ajouter l'indicateur dans les sélecteurs.

D. Maintenir un mode de représentation

Afin d'ajouter un mode de représentation, il faut connaître la structure des graphiques dans l'application, pour cela, se référer à la partie du récapitulatif des variables de stockage de données.

1) Ajouter la variable globale du graphique

Tout d'abord, il vous faut créer la variable liée à votre graphique. Rendez-vous dans le fichier *R/Datas/datasForServerFct.R* et ajoutez-y votre variable en suivant le modèle des autres. Il est conseillé de faire une liste *list(isPlot, plot, titre)* si il ne peut y avoir qu'un seul graphique de la sorte affiché et *list(nbPlot, plot1, ... plotn, titre1, ..., titren)* si il peut y avoir n figures en même temps.

```

piePlot <- list(0,NULL,NULL,"","");
histoPlot <- list(FALSE,NULL,"");

```

2) Créer la fonction permettant de tracer le graphique

La deuxième étape consiste à créer le graphique automatiquement en fonction d'un indicateur sélectionné, du pôle sélectionné, d'une année, etc. Pour cela, créez une fonction dans un fichier à part en suivant le modèle des autres fonctions comme *R/Datas/afficher_hist.R* (1 seul graphique) ou *R/Datas/afficher_pie.R* (2 graphiques selon le/les pôles sélectionnés). Vous pouvez mettre dans cette fonction tous les arguments nécessaires à la détermination du graphique à afficher.

Ensuite, ajoutez un appel de cette fonction à la fonction *datasForServerFct* du fichier *R/Datas/datasForServerFct.R* comme suit :

```

print("Création d'un histogramme...");
histoPlot <- afficher_hist(groupe,pole,taxo,type);

```

N'oubliez pas d'appeler ce script dans le fichier *R/loadScripts.R* comme pour les autres fonctions :

```
getwd() %>%
  paste0("/R/Datas/afficher_carte.R") %>%
  source()
```

3) Créer les div liées à votre graphique

Une fois votre fonction créée, avant de l'appeler, il faut préparer les outputs. Tout d'abord, allez dans le fichier *R/Datas/createDivDatas.R* et créez une div qui contiendra votre graphique comme suit (attention, il ne faut pas de ";" à la fin du fichier) :

```
divHist <- div(id="histogramme");
```

Ensuite, insérez votre div dans la "zone des datas viz".

```
# Zone des data viz
divDataViz <- div(id = "dataViz",
                    divDataSumUp,
                    divHist,
                    divPie,
                    divBar,
                    divFournProd)
```

4) Utiliser la fonction de création de graphique dans la div associée

L'avant dernière étape consiste à insérer votre graphique dans la page. Il vous faut alors aller dans la fonction *R/Datas/dispDatasForServerFct.R* et ajouter le graphique à l'ui en fonction de si il est présent ou non.

Si un seul graphique ne peut être affiché qu'une seule fois, rajouter des lignes de codes suivant le modèle suivant. Dans ce cas, "histWithTitle" est un nom de div au choix, **"histogramme" doit être le nom de la div créée dans l'étape 3. Le nom donné dans plotlyOutput, "hist" ici, doit être le même nom que le 1er output "output\$hist".** Le deuxième output servira à la dernière étape pour la popup.

```
# Actualisation de l'histogramme
removeUI(selector = "#histWithTitle");
if (histoPlot[[1]]) {
  insertUI(selector = "#histogramme",
           ui = div(id = "histWithTitle",
                     class = "graphWithTitle",
                     tags$p(histoPlot[[3]]),
                     plotlyOutput('hist')));
  output$hist <- renderPlotly({histoPlot[[2]]});
  output$histCopy <- renderPlotly({
    ggplotly(histoPlot[[2]]) %>% layout(height = 500, width = 1100)
  });
}
```

Si un seul graphique peut être affiché n fois, rajouter des lignes de codes suivant le modèle suivant. Dans ce cas, et autant de fois qu'il y a de graphiques possibles dans votre variable globale, effectuer un test if de cette manière. "pieNWithTitle" est un nom de div au choix, **"pieChart" doit être le nom de la div créée dans l'étape 3. Le nom**

donné dans chaque `plotlyOutput`, “`pieN`” ici, doit être le même nom que le 1er output du if en question, “`output$pieN`”. Le deuxième output servira à la dernière étape pour la popup.

```
# Actualisation du pie chart
removeUI(selector = "#pie1WithTitle");
removeUI(selector = "#pie2WithTitle");
if (piePlot[[1]] >= 1) {
  insertUI(selector = "#pieChart",
    ui = div(id = "pie1WithTitle",
      class = "graphWithTitle",
      tags$p(piePlot[[4]]),
      plotlyOutput('pie1')));
  output$pie1 <- renderPlotly({piePlot[[2]]});
  output$pie1Copy <- renderPlotly({
    ggplotly(piePlot[[2]]) %>% layout(height = 500, width = 1100)
  });
}
if (piePlot[[1]] >= 2) {
  insertUI(selector = "#pieChart",
    ui = div(id = "pie2WithTitle",
      class = "graphWithTitle",
      tags$p(piePlot[[5]]),
      plotlyOutput('pie2')));
  output$pie2 <- renderPlotly({piePlot[[3]]});
  output$pie2Copy <- renderPlotly({
    ggplotly(piePlot[[3]]) %>% layout(height = 500, width = 1100)
  });
}
```

5) Création d'un popup pour votre graphique

La dernière étape de l'ajout d'un type de graphique est la création du popup associé. Tout d'abord, rendez-vous dans le fichier `www/JavaScript/idDataVizAccueilFromHoverProcess.js`. Après la liste des rN, ajoutez-y le votre en remplaçant avec l'id de votre/vos outputs de graphique (pour chaque output possible). Par exemple pour les pie chart, on ajoutera les lignes suivantes :

```
let r2 = event.target.closest("#pie1");
let r3 = event.target.closest("#pie2");
```

Ensuite, ajoutez le ou les tests if à la suite des précédents, où vous initialiserez la valeur de `input$idDataViz` avec l'id de l'output pour s'assurer qu'il est unique.

```
else if (r3!==null) {
  /*On modifie un input R qui va être écouté par un ObserverEvent*/
  Shiny.setInputValue('idDataViz', "pie2", {priority: "event"});
}
```

Enfin, allez dans le fichier R/Server/popUpDataVizFct.R et ajoutez un test if pour chaque graphique suivant le modèle suivant, sachant que le la valeur de l'input dans le test if doit être celui envoyé par JavaScript. dans le tag\$p, le deuxième argument doit être le titre du graphique, c'est à dire le titreN pour le plotN. Enfin, dans le plotlyOutput vous devez mettre l'id du second output rempli dans l'étape 4, celui de la copie du graphique.

```

else if (input$IdDataViz=="piel"){
  insertUI(selector = "#popUpDataViz", ui = div(id = "graphCopy",
                                                div(id = "buttonCloseDiv",
                                                    tags$p(id = "titrePopup",
                                                       piePlot[[4]]),
                                                    actionButton(inputId = "closePopupButton",
                                                               icon("window-close"))),
                                                plotlyOutput('pielCopy')));
}

```

Après avoir fait toutes ces étapes, vous avez un nouveau graphique opérationnel qui peut être personnalisé dans sa fonction créée à l'étape 2 afin qu'il réponde comme vous le souhaitez à vos indicateurs, pôles ,déclinaisons, etc.

E. Maintenir une déclinaison et ses groupes associés

a. Ajouter une déclinaison dans les sélecteurs

Afin d'ajouter une déclinaison dans le sélecteur associé, il faut tout d'abord comprendre la logique de l'implémentation de ces derniers dans l'application. Pour cela, référez vous à la partie du récapitulatif des variables de stockage de données.

1) Remplissage du tableau récapitulant les déclinaisons possibles

Ensuite, rendez-vous dans le fichier *R/UI/body/createListIndicators.R* et cherchez la création des listes suivantes :

```

listDeclinaisonIndName <- c("Général",
                            "Pôles",
                            "Taxonomie",
                            "Par fournisseur",
                            "Par producteur");
listDeclinaisonInd <- c("general",
                        "poles",
                        "taxo",
                        "fournisseur",
                        "producteur");

```

À la première liste, rajoutez le nom que vous souhaitez donner à votre déclinaison et à la deuxième, ajoutez-y une “valeur” associée (un string simplifié).

2) Mise à jour des indicateurs et de leur dépendance à cette déclinaison

Par la suite, rendez vous à la liste suivante et ajoutez-y “is<Delcinaison>” à la fin selon un nom adéquat.

```

listColumns <- c("indName", "ind", "isGlobal", "isExpert",
                 "typeInd",
                 "isPoles", "isTaxo", "isFournisseur", "isProducteur",
                 "isAnnees")

```

Enfin, il vous faut modifier la structure de vos indicateurs et leur implication dans le code. Cherchez donc la création de la liste d'indicateurs qui suit. Ajoutez à chaque indicateur le booléen correspondant à la bonne position (y compris pour le premier élément).

```
# Liste associée aux indicateurs
tabIndicators <- data.frame(indName = "Nombre de données",
                             ind = "données",
                             isGlobal = TRUE,
                             isExpert = TRUE,
                             typeInd = "Indicateur de connaissance",
                             isPoles = TRUE,
                             isTaxo = TRUE,
                             isFournisseur = TRUE,
                             isProducteur = TRUE,
                             isAnnees = TRUE)

1er indicateur

# indName, ind, isGlobal, typeInd, isPoles, isTaxo, isFournisseur, isProducteur, isAnnees
newRow <- list("Nombre d'espèces", "especes", TRUE, TRUE, "Indicateur de connaissance", TRUE, TRUE, FALSE, FALSE, TRUE);
tabIndicators <- rbind(tabIndicators,newRow);

Tous les autres indicateurs
```

Vous pouvez aussi ajouter votre déclinaison à la liste suivante pour ceux qui reprendront le code.

```
# indName, ind, isGlobal, typeInd, isPoles, isTaxo, isFournisseur, isProducteur, isAnnees
```

3) Remplissage du sélecteur avec le nouvel indicateur

Comme dit précédemment, il faut prendre en compte cette déclinaison lors de la modification d'indicateur, c'est pourquoi il faut se rendre dans le fichier *modifySelectors.R*, à chaque création de la liste “listDecli”,

```
# Remplissage du selector de déclinaison
listDecli <- c(TRUE);

# Vérification que la déclinaison est légale
listDecli <- c(TRUE);
```

Ajoutez-y votre déclinaison **à la même position qu'à la première étape** selon le modèle suivant :

```
listDecli <- append(listDecli, findIndicateurInfoByNum(numInd, "isProducteur"));
```

Votre type d'indicateur est maintenant présent dans votre sélecteur dans la page expert.

b. Ajouter une taxonomie dans les sélecteurs

Afin d'ajouter une taxonomie dans le sélecteur associé, il faut tout d'abord comprendre la logique de l'implémentation de ces derniers dans l'application. Pour cela, référez vous à la partie du récapitulatif des variables de stockage de données.

Ensuite, rendez-vous dans le fichier *R/UI/body/createListIndicators.R* et cherchez la balise suivante :

```
# ---!!! DERNIERE LIGNE GROUPE !!!--- #
```

Au niveau de cette balise, ajoutez deux lignes selon le modèle suivant (défini dans la structure du stockage de données) :

```
newRow <- list("Lichens", TRUE, FALSE, FALSE);  
tabGroupe <- rbind(tabGroupe,newRow);
```

Votre taxonomie est maintenant présente dans votre sélecteur si la déclinaison taxonomie est active en experte et que le pôle le permet.

F. Réimplémentation de la page administrateur

Pour la réimplémentation de la page administrateur, il suffit de décommenter certains codes qui sont les suivants :

La création du mot de passe *mdpAdmin.txt* (il faut également l'ajouter dans le dossier parents, au même niveau que *app.R*)

```
# ----- TOUTES LES INITIALISATIONS ----- #  
# ----- INITIALISATION DES MOTS DE PASSE ----- #  
# mdpAdmin = read_file("mdpAdmin.txt")  
# gestion des caractères \\ \  
# mdpAdmin <- gsub("\\\\", "", mdpAdmin)
```

Dans *R/Server/createServer.R*

La création des boutons liés à la page

```
# Boutons de navigation actifs  
accueilButton <- actionButton(inputId = "accueilButton", label = "ACCUEIL")  
globalButton <- actionButton(inputId = "globalButton", label = "GLOBAL")  
expertButton <- actionButton(inputId = "expertButton", label = "EXPERT")  
# adminButton <- actionButton(inputId = "adminButton", label = "ADMIN")  
  
# Boutons de navigation passifs  
passiveAccueilButton <- actionButton(inputId = "passiveButton", label = "ACCUEIL")  
passiveGlobalButton <- actionButton(inputId = "passiveButton", label = "GLOBAL")  
passiveExpertButton <- actionButton(inputId = "passiveButton", label = "EXPERT")  
# passiveAdminButton <- actionButton(inputId = "passiveButton", label = "ADMIN")
```

Dans *R/UI/head/createNavButtons.R*

L'ajout des boutons dans le bandeau du haut

```
# div qui contiennent les boutons de navigation
divAccueilButton <- div(id = "divAccueilButton",
                        passiveAccueilButton)
divGlobalButton <- div(id = "divGlobalButton",
                        globalButton)
divExpertButton <- div(id = "divExpertButton",
                        expertButton)
# divAdminButton <- div(id = "divAdminButton",
#                         adminButton)

# Ensemble des boutons de navigation
ulNavigation <- tags$ul(id = "navigation",
                        divAccueilButton,
                        divGlobalButton,
                        divExpertButton) #,
# divAdminButton
```

La *R/UI/head/createUI.R*

Après modification de ces trois fichiers, la page admin est de nouveau disponible avec une navigation interne mais sans fonctionnalités.

FIN

Annexes

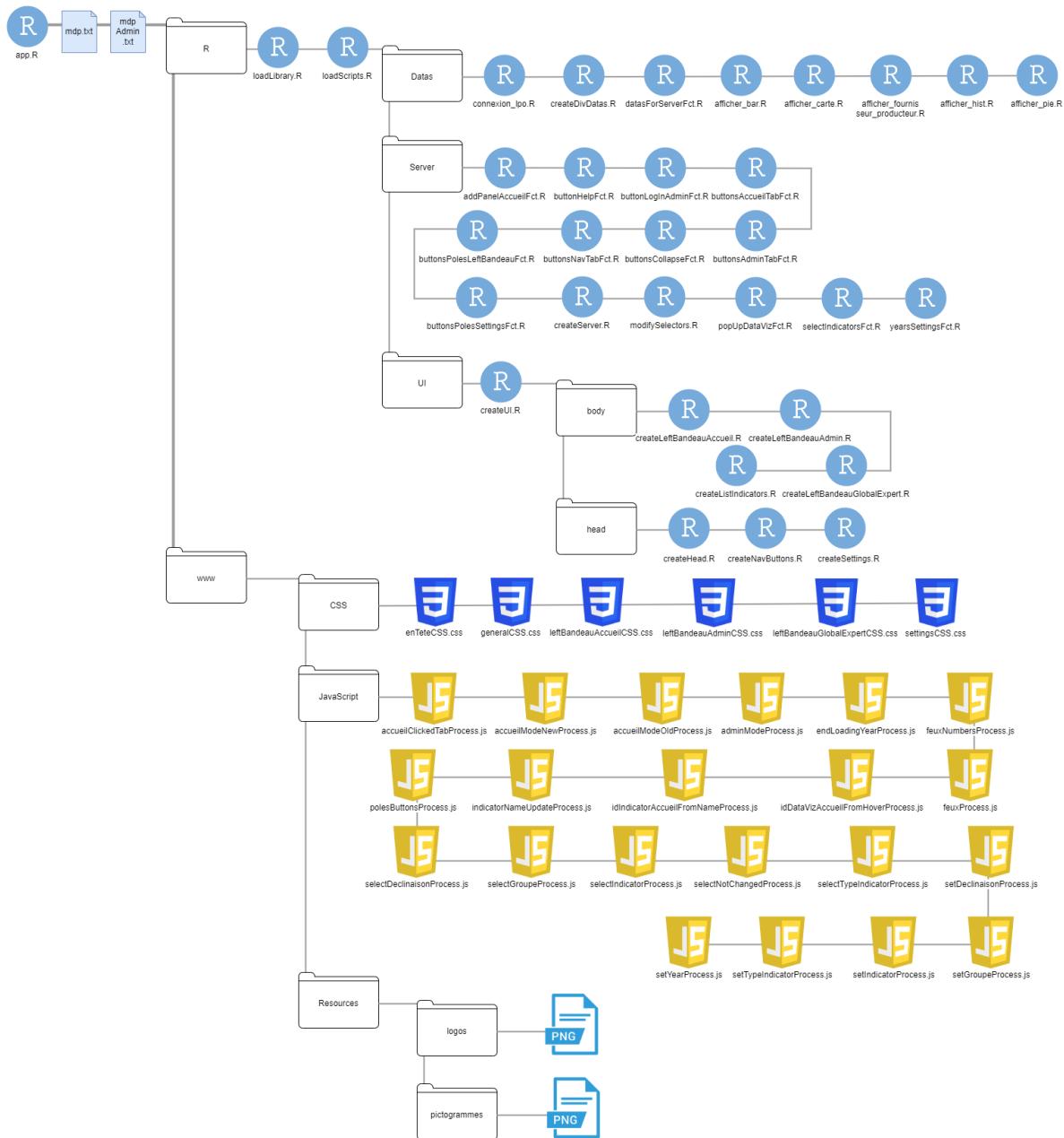


Figure 1 - Architecture du code source

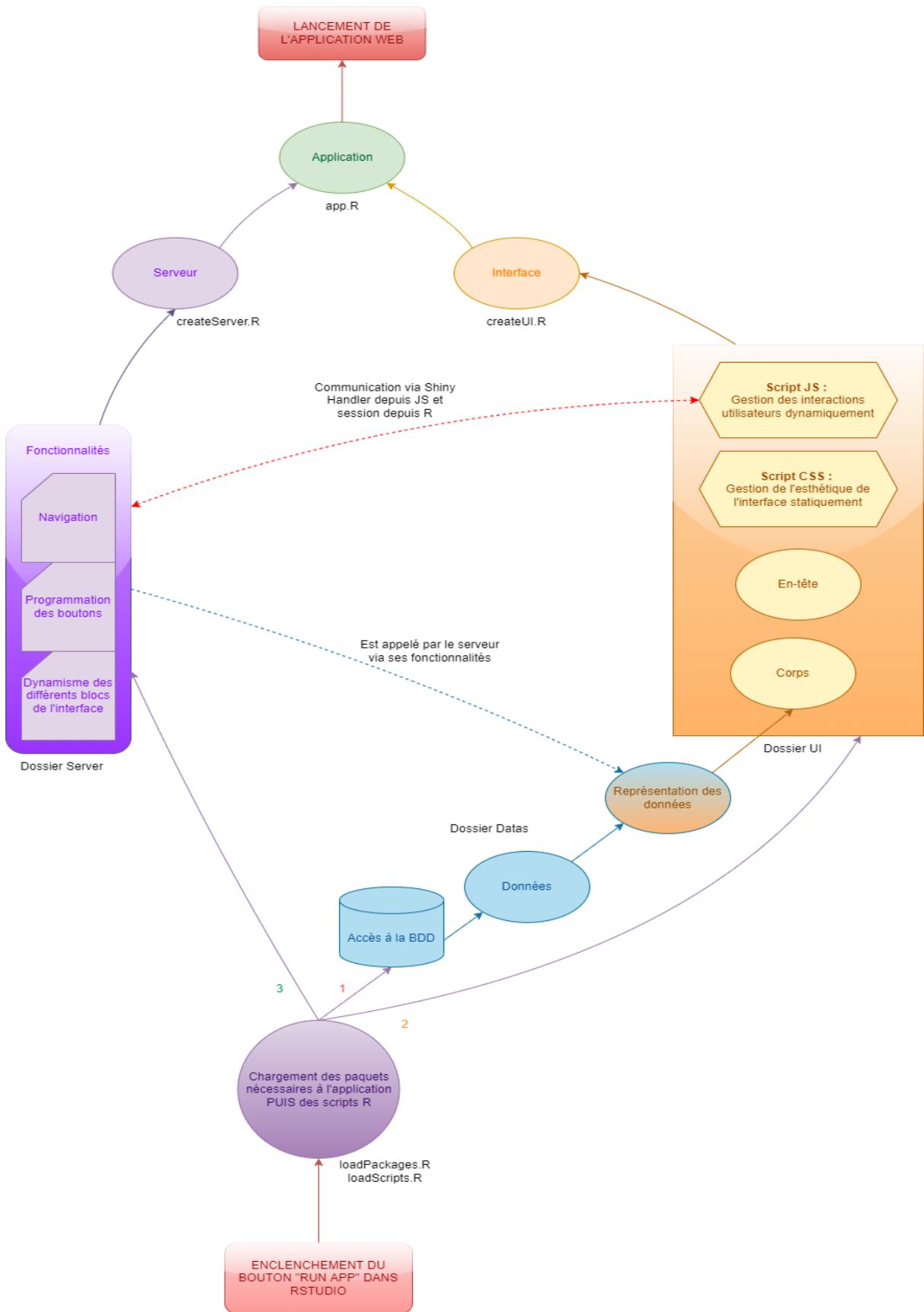


Figure 2 - Logigramme du lancement de l'application