Work Package 5 EU-CEG data and enhanced laboratory capacity for regulatory purposes

Dashboard and how-to guide to analyse EU CEG data at national scale

Authors: ANSES
January 2024

Doc. Ref. N°: D5.4
Type: Document (R)
Dissemination: Public

JATC 2 JOINT ACTION ON TOBACCO CONTROL

Co-funded by the European Union's Health Programme under Grant Agreement n°: 101035968 - JA-01-2020 - HP-JA-2020 / HP-JA-2020-2

Version	Date	Authors	Comments
1	29 Jan. 2024	ANSES	Submitted version

Table of contents:

Introduction

The European Union (EU) Common Entry Gate (EUCEG) is a digital front-office mechanism by which manufacturers and importers of tobacco and related products submit key information about the products they intend to market in one or several EU Member States (MS)[1]. The submitters report such information in an electronic extensible mark-up language (XML) format implemented for this purpose[2].

On the back-office side, MS Competent Authorities (MSCA) can access to the reported information through a secured web interface called the MS Reporting Tool (MS-Rep). Only staff from MSCA who are duly authorized to log in MS-Rep by each relevant national administrator can access to the corresponding national data warehouse hosted on European Commission servers.

MS-Rep consists in a web interface to search, browse, view, query and export data from the manufacturers' electronic submissions about their products.

The JATC2D5.2 deliverable describes a procedure to build local SQLite[3] databases (DB) of EUCEG data for both tobacco (TP) and e-cigarette (EC) products. This step of getting an existing local database is a prerequisite prior any installation of the solution presented here.

This document describes a procedure to install and run an open source tool to analyse EUCEG data:

EU-CEG Viewer.

It is based on the statistical software R and its package Shiny[5].

1. Scope and limits

This document addresses the question of EU-CEG data analysis at national scale, which means that it is intended for MSCA dealing with the submissions they have received and integrated into a local DB according to procedure described in JATC2D5.2 deliverable.

There is one **EUCEG Viewer** dashboard for EC products and one for TP products: they are dependent on the respective data structures of **EC.db** and **TP.db.**

The procedure has been developed and tested under **Microsoft Windows 10 (64 bits)** as an operating system (OS), which is supposed to be currently the most common environment used by MSCA. Some adjustments to other OS like MacOS or Linux would be needed.

As of the date of this report, the external software used by this procedure are open source, free to use for the intended purpose and available for download from Internet with a history of previous versions. Using a more recent version of those software in the future may cause the procedure not to work properly.

The scripts developed for this procedure (see source codes) are covered by the European Union Public Licence (EUPL-1.2 or later)[6].

1 https://health.ec.europa.eu/eu-common-entry-gate-eu-ceg_en

2 https://health.ec.europa.eu/eu-common-entry-gate-eu-ceg/download-section_en

3 https://www.sqlite.org/

4 https://cran.r-project.org/

5 https://shiny.posit.co/

6 https://joinup.ec.europa.eu/collection/eupl/eupl-text-eupl-12

The procedure does not deal with data safety issues related to confidentiality or trade secrets contained in the EU-CEG DB files handled by EUCEG Viewer dashboard. It is intended to be run on a local machine with appropriate data protection relevant to each MSCA policy: password protection for the user session, controlled access to the premises and/or encrypted hard disk for a laptop… Except for downloading and installing the external software once and the EU-CEG XML files needed to update the DB files, the procedure does not need an Internet access to work. It is recommended that all files containing potentially confidential data are stored on a local machine with appropriate protection and not on a server with multiple accesses.

Both EUCEG Viewer dashboard and DB file are intended for a single-user access: should many staff need to work on the same data, consider to distribute several instance copies of the same DB on the different protected local machines instead of sharing a unique DB file on a shared folder of the network. As a consequence, EUCEG Viewer is designed to run on a local machine and not on a server.

A minimum knowledge is required about how to follow instructions to install a software, manage files and folders, and edit any file with a text editor (eg. Notepad) to customize some command line parameters if needed. Some users may need assistance with IT skilled staff to go through this procedure with their own data.

The scripts and software can be installed and run on a client machine without administrator's rights.

2. Instructions for installation and use

The **external software** is provided with the scripts through **eucegviewer.zip** archive available from the **JATC2 CircaBC repository**[7].

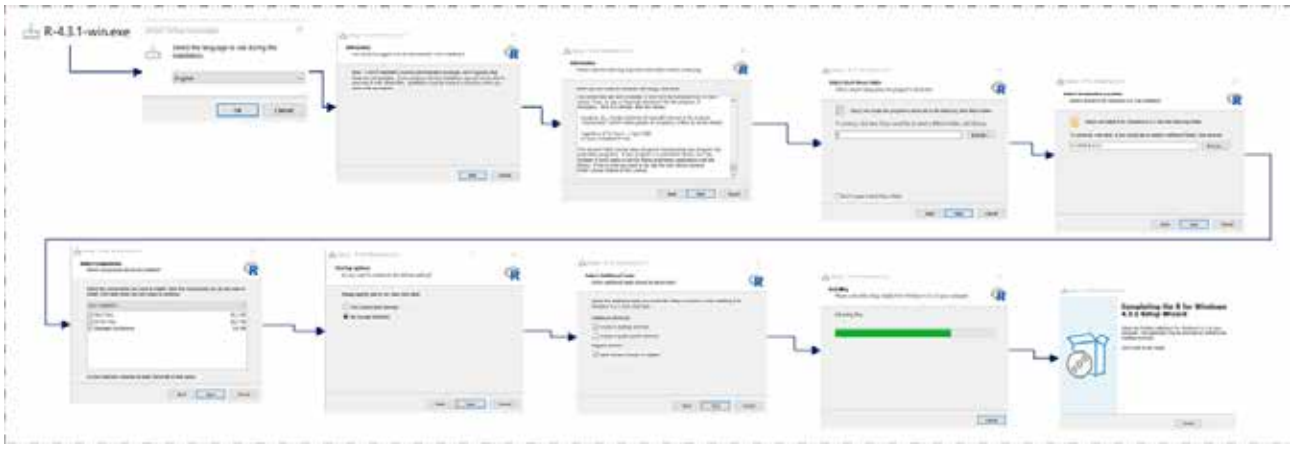
Please follow the next steps to install EUCEG Viewer.

Installation

0 Make sure R is installed

If not, download R from https://cran.r-project.org/bin/windows/base/old/4.3.1/R-4.3.1-win.exe

Then, launch the R-4.3.1-win.exe and follow the instructions

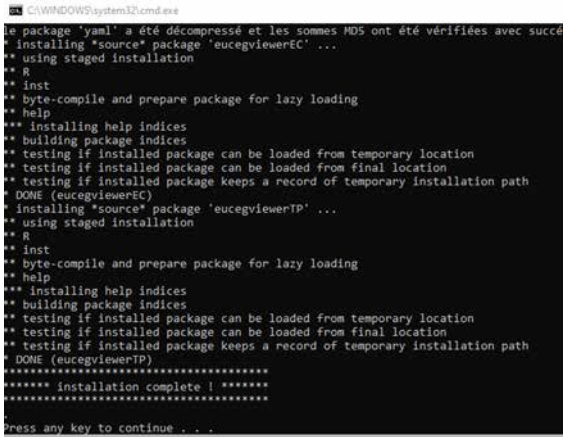


Install R locally with default options in an user directory (eg. C:\APPS\R-4.3.1\)

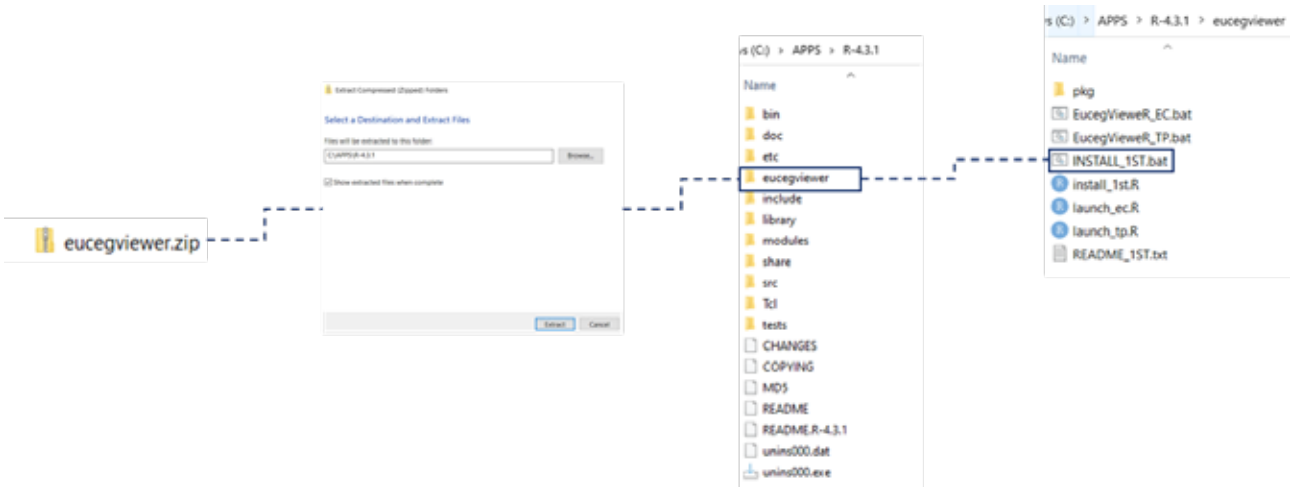
7 https://circabc.europa.eu/ui/group/0fa614cf-8769-4f8d-b81e-e5cd7c837185

Installation

1 Download eucegviewer.zip from CircaBC



Windows command prompt



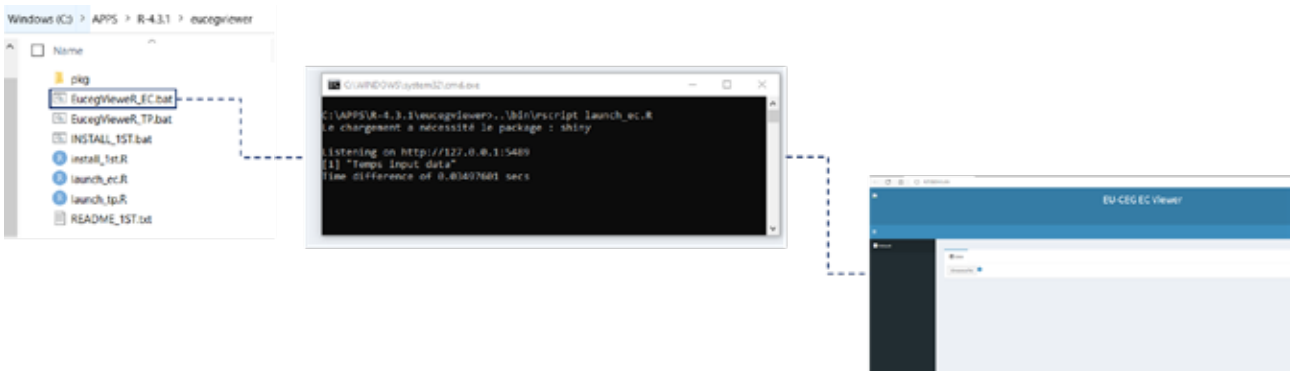
2 Unzip it into the R folder (eg. C:\APPS\R-4.3.1\eucegviewer\)

3 Run INSTALL_1ST.bat ONCE to install the dashboards.

Launching EU-CEG Viewer

1 Run either EucegVieweR_EC.bat or EucegVieweR_TP.bat to launch the viewer for EC or TP products.

The Windows command prompt will open and the dashboard EU-CEG Viewer will open in a browser window.



Database selection

0 When the application is launched, a window will open (in a browser).

1 Click on “Choose the database”. It will open a file browser.

2 In the file browser, choose your SQLite database.

The extension should be “.db” (currently EC.db or TP.db).

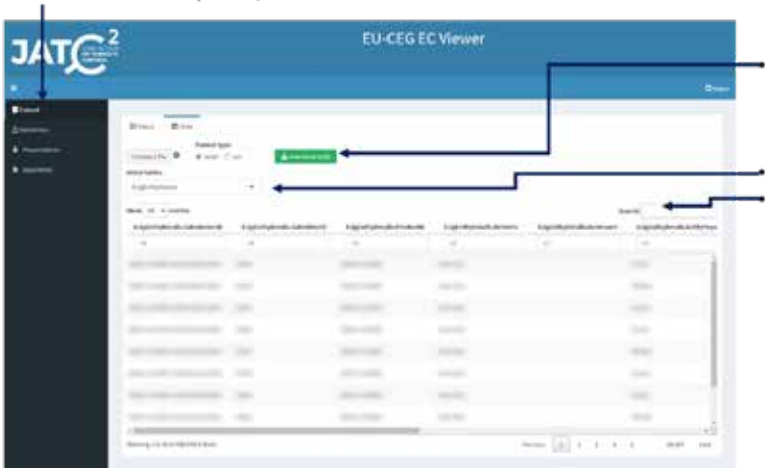
3 After the loading message, you will be noticed when the file will be imported

You might have to wait for a minute before going further because of data preparation the app does in background.

Application tab - Dataset

The dataset tab displays raw data as data frames.

Once the data are imported, the 4 tabs are accessible.



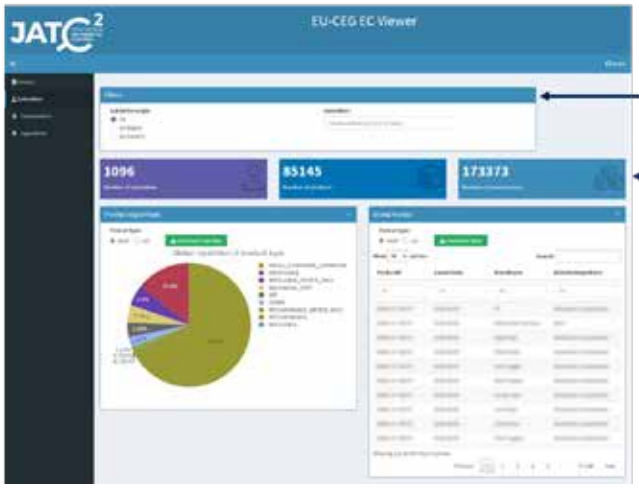
The data can also be downloaded using the "Download data" button, the format can be chosen between xlxs or csv. If filters are applied, filtered data will be download.

You can select the table in the drop down list.

A word can be searched in each cells of the table, or the data frame can be filtered with filters placed above each column (recommended for a faster and more precise execution).

Application tab - Submitters

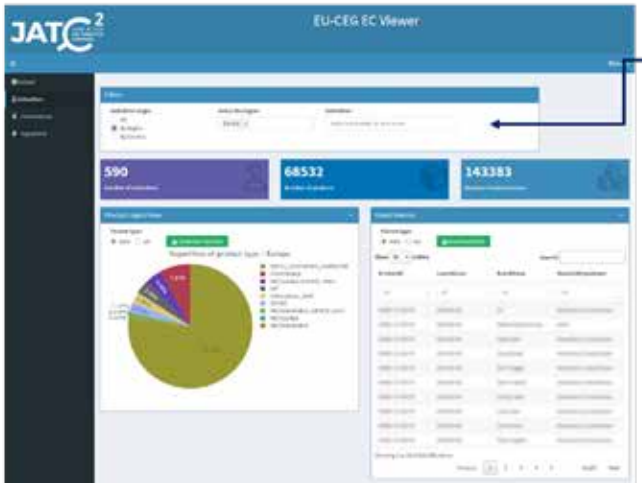
The submitter tab displays global information about product submitters.



There are global filters (all submitters, or submitters by region and country), but you can choose one or many specific submitters to see their informations.

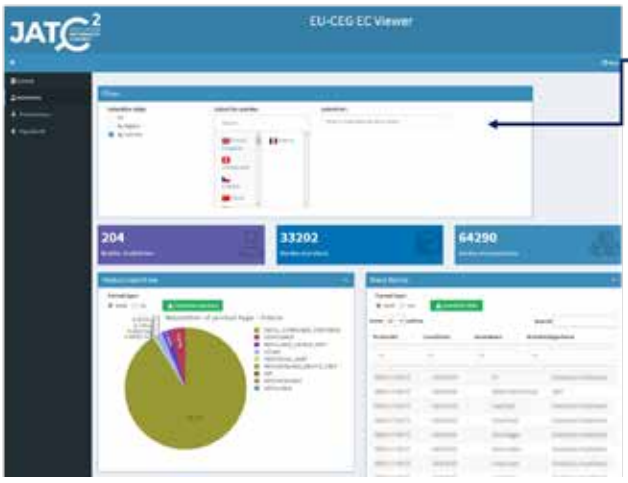
Note : a presentation is the declination of a product. So, one product can be sold under different presentations, but a presentation is always related to one product. Sometimes, presentations from different products can have the same name.

Application tab - Submitters



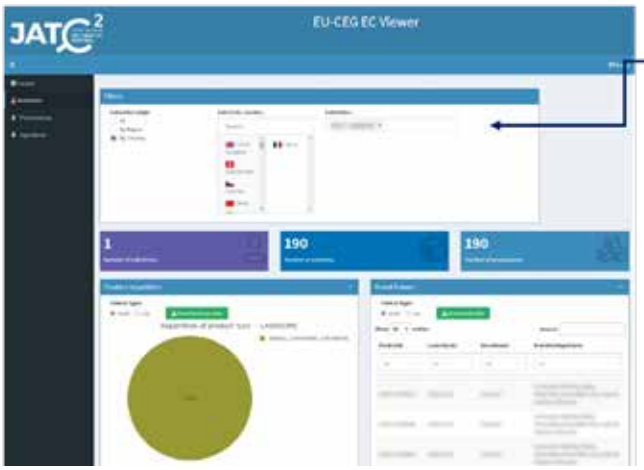
If you choose submitter origin by region, you will be able to choose a specific region to look at. The list of submitters will also adapt to the location you chose and in this case, only European submitters will be available in the submitter field

Application tab - Submitters



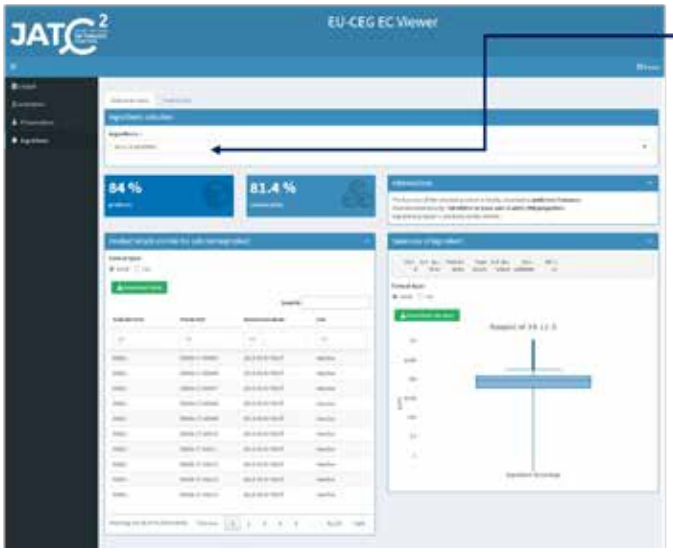
If you choose submitter origin by country, you will be able to select only submitter from this country. Since no submitter is chosen, the information will display for the country you picked in the list.

Application tab - Submitters



Finally, as soon as you choose a submitter, the page will adapt to the information for this specific submitter.

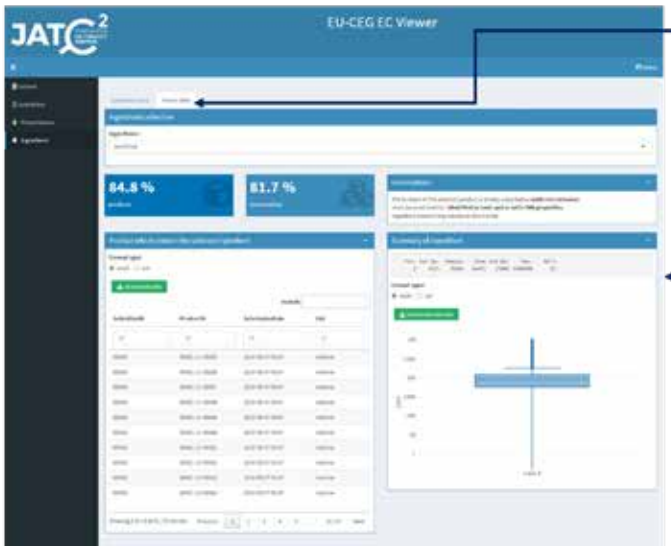
Application tab - Ingredients



You can seek for an ingredient, but you will have to choose between all the ways it has been declared.

For example "nicotine" have different types of declaration for the same molecule.

Application tab - Ingredients



If you want to get information about the ingredients without taking into account the way they have been declared, they have been aggregated by Parent names. So, you will be able to browse for an ingredient by its generic name (and not the CAS number).

- When the ingredient is selected, you get the list of products that contain such ingredient, and global statistics about the global ingredient repatriation. Values are the related proportion of the ingredient compared to other in each product, and they are in ppm.

3. Source code: <EC> app_server.R

```
# app_server.R version 23-01-2024
#
# Copyright (c) Thomas El Khilali - euceg@anses.fr 2022-2023 (JATC2-WP5)
# Except specific files which bear a different mention, this programme is licensed under the EUPL-
1.2 or later
# You may obtain a copy of the license at https://joinup.ec.europa.eu/collection/eupl/eupl-text-
eupl-12
#
# This activity has received funding from the European Union's Health Program (2014-2020) under
grant agreement N°101035968 (JA-01-2020 - HP-JA-2020 - HP-JA-2020-2).
# The content of this document represents the views of the author only and is his/her sole
responsibility; it cannot be considered to reflect the views of the European Commission and/or the
European Health and Digital Executive
# Agency (HaDEA) or any other body of the European Union. The European Commission and the Agency do
not accept any responsibility for use that may be made of the information it contains.


# Increase maximum file size
options(shiny.maxRequestSize = 4000*1024^2)

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

  # when browser is closed, stop the terminal
  session$onSessionEnded(stopApp)

  # %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  # Data Importation
  # %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  mod_inputData_server(id = "inputData_1", r=r)
  # %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  # Submitters
  # %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  output$submittersMenu <- renderMenu({
    req(r$dataBase)
    menuItem(text = "Submitters", tabName = "submitters", icon = icon("user"))
  })

  output$presentationMenu <- renderMenu({
    req(r$dataBase)
    menuItem(text = "Presentations", tabName = "presentations", icon = icon("baby-formula", lib =
"glyphicon"))
  })

  output$ingredientsMenu <- renderMenu({
    req(r$dataBase)
    menuItem(text = "Ingredients", tabName = "ingredients", icon = icon("tint", lib = "glyphicon"))
  })

  mod_submitters_server(id = "submitters_1", r=r)
  mod_presentation_server(id = "presentation_1", r=r)
  mod_ingredients_server("ingredients_1", r=r)

}
```

4. Source code: <EC> app_ui.R

```
# app_ui.R version 23-01-2024
#
# Copyright (c) Thomas El Khilali - euceg@anses.fr 2022-2023 (JATC2-WP5)
# Except specific files which bear a different mention, this programme is licensed under the EUPL-
1.2 or later
# You may obtain a copy of the license at https://joinup.ec.europa.eu/collection/eupl/eupl-text-
eupl-12
#
# This activity has received funding from the European Union's Health Program (2014-2020) under
grant agreement N°101035968 (JA-01-2020 - HP-JA-2020 - HP-JA-2020-2).
# The content of this document represents the views of the author only and is his/her sole
responsibility; it cannot be considered to reflect the views of the European Commission and/or the
European Health and Digital Executive
# Agency (HaDEA) or any other body of the European Union. The European Commission and the Agency do
not accept any responsibility for use that may be made of the information it contains.


#' The application User-Interface
#'
#' @param request Internal parameter for `{shiny}`.
#'     DO NOT REMOVE.
#' @import shiny
#' @import shinythemes
#' @import shinydashboard
#'
#' @noRd
app_ui <- function(request) {
  tagList(
    # Leave this function for adding external resources
    golem_add_external_resources(),
    # shinyjs::useShinyjs(),

    # To use icon from font awesome
    # tags$script(src = "https://kit.fontawesome.com/4afa144779.js"), #
https://fontawesome.com/v4/icons/

    shinydashboard::dashboardPage(
      title = "EU-CEG Viewer",
      # skin = "purple",
      ## Header with logo and App title
      header = dashboardHeader(
        title = HTML("<div style = 'vertical-align:middle'>
                          <img src = 'www/jatc2.png' align = 'left' height = '100px'>
                          <h1>EU-CEG EC Viewer</h1>
                      </div>"
                    ),
        titleWidth = "92%"
        # tags$li(class = "dropdown",
tags$a(href="https://github.com/ThomasElKhilali/dashboard_anses", icon("github"), "Repos"))
      ),

      dashboardSidebar(
        sidebarMenu(
          id = "sidebar",
          menuItem(text = "Dataset", tabName = "data", icon = icon("database")),
          # menuItem(text = "Submitters", tabName = "submitters", icon = icon("user"))
          menuItemOutput("submittersMenu"),
          # menuItem(text = "Vizualisation", tabName = "viz", icon = icon("chart-line"))
          menuItemOutput("presentationMenu"),
          menuItemOutput("ingredientsMenu")
        )
      ),


      dashboardBody(
        # intern function for css header style
        headerCssStyle(),
        tabItems(
          # Onglet Data :
          mod_inputData_ui("inputData_1"),
          # Onglet Submission :
          mod_submitters_ui("submitters_1"),
          # Onglet presentation
          mod_presentation_ui("presentation_1"),
```

```
          # Onglet Ingredient
          mod_ingredients_ui("ingredients_1")

        )
      )
    )
  )
}

#' Add external Resources to the Application
#'
#' This function is internally used to add external
#' resources inside the Shiny application.
#'
#' @import shiny
#' @importFrom golem add_resource_path activate_js favicon bundle_resources
#' @noRd
golem_add_external_resources <- function() {

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

  tags$head(
    favicon(),
    bundle_resources(
      path = app_sys("app/www"),
      app_title = "eucegviewerEC"
    )

    # Add here other external resources
    # for example, you can add shinyalert::useShinyalert()
  )
}
```

5. Source code: <EC> fct_headerCssStyle.R

```
# fct_headerCssStyle.R version 23-01-2024
#
# Copyright (c) Thomas El Khilali - euceg@anses.fr 2022-2023 (JATC2-WP5)
# Except specific files which bear a different mention, this programme is licensed under the EUPL-1.2 or later
# You may obtain a copy of the license at https://joinup.ec.europa.eu/collection/eupl/eupl-text-eupl-12
#
# This activity has received funding from the European Union's Health Program (2014-2020) under grant agreement N°101035968 (JA-01-2020 - HP-JA-2020 - HP-JA-2020-2).
# The content of this document represents the views of the author only and is his/her sole responsibility; it cannot be considered to reflect the views of the European Commission and/or the European Health and Digital Executive
# Agency (HaDEA) or any other body of the European Union. The European Commission and the Agency do not accept any responsibility for use that may be made of the information it contains.


#' Header Css style
#' @noRd
headerCssStyle <- function(){

  tags$style(type="text/css", "
          /* https://jonkatz2.github.io/2018/06/22/Image-In-Shinydashboard-Header
          https://stackoverflow.com/questions/48978648/align-header-elements-in-shiny-dashboard */

          /*    Move everything below the header */
              .content-wrapper {
                  margin-top: 50px;
              }
              .content {
                  padding-top: 60px;
              }
          /*  Format the title/subtitle text */
              .title-box {
                  position: absolute;
                  text-align: center;
                  top: 50%;
                  left: 50%;
                  transform:translate(-50%, -50%);
              }
              @media (max-width: 590px) {
                  .title-box {
                      position: absolute;
                      text-align: center;
                      top: 10%;
                      left: 10%;
                      transform:translate(-5%, -5%);
                  }
              }

          /*    Make the image taller */
              .main-header .logo {
                  height: 125px;
              }
          /*    Override the default media-specific settings */
              @media (max-width: 5000px) {
                  .main-header {
                      padding: 0 0;
                      position: relative;
                  }
                  .main-header .logo,
                  .main-header .navbar {
                      width: 100%;
                      float: none;
                  }
                  .main-header .navbar {
                      margin: 0;
                  }
                  .main-header .navbar-custom-menu {
                      float: right;
                  }
              }
          /*    Move the sidebar down */
              .main-sidebar {
```

```
                position: absolute;
            }
            .left-side, .main-sidebar {
                padding-top: 175px;
            }"
    )
}
```

6. Source code: <EC> mod_ingredients.R

```
# mod_ingredients.R version 23-01-2024
#
# Copyright (c) Thomas El Khilali - euceg@anses.fr 2022-2023 (JATC2-WP5)
# Except specific files which bear a different mention, this programme is licensed under the EUPL-
1.2 or later
# You may obtain a copy of the license at https://joinup.ec.europa.eu/collection/eupl/eupl-text-
eupl-12
#
# This activity has received funding from the European Union's Health Program (2014-2020) under
grant agreement N°101035968 (JA-01-2020 - HP-JA-2020 - HP-JA-2020-2).
# The content of this document represents the views of the author only and is his/her sole
responsibility; it cannot be considered to reflect the views of the European Commission and/or the
European Health and Digital Executive
# Agency (HaDEA) or any other body of the European Union. The European Commission and the Agency do
not accept any responsibility for use that may be made of the information it contains.


#' ingredients UI Function
#'
#' @description A shiny Module.
#'
#' @param id,input,output,session Internal parameters for {shiny}.
#'
#' @noRd
#'
#' @importFrom shiny NS tagList
mod_ingredients_ui <- function(id){
  ns <- NS(id)
  tabItem(tabName = "ingredients",

          tabsetPanel(type = "tabs",
                      tabPanel("Substance stats",
                               fluidRow(
                                 box(title = "Ingredients selection", solidHeader = TRUE, status =
"primary", width = 12,
                                     selectizeInput(
                                       inputId = ns("ingredientsList"),
                                       label = "Ingredients :",
                                       multiple = FALSE,
                                       choices = NULL,
                                     )
                                 )
                               ),
                               fluidRow(
                                 # textOutput(ns("propProd")),
                                 # textOutput(ns("propPres")),
                                 shinydashboard::valueBoxOutput(ns("propProd"), width = 3),
                                 shinydashboard::valueBoxOutput(ns("propPres"), width = 3),
                                 ## Statut :
                                 box(title = "Informations", width = 6,
                                     solidHeader = TRUE, status = "primary", collapsible = TRUE,
                                     htmlOutput(ns("fonction")),
                                     htmlOutput(ns("toxicity")),
                                     textOutput(ns("onMarket"))
                                 ),

                               ),

                               fluidRow(
                                 box(title = "Product which contain the selected ingredient",
                                     solidHeader = TRUE, status = "primary", collapsible = TRUE,
                                     radioButtons(ns("typeproductOnMarket"), "Format type:",
                                                  choices = c(excel = "xlsx", csv = "csv"), inline =
TRUE),
                                     downloadButton(ns("dwdproductOnMarket"), "Download table",
                                                    style = "color: #fff; background-color: #27ae60;
border-color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                                     DTOutput(ns("productOnMarket"))
                                 ),
                                 box(title = "Summary of ingredient", collapsible = TRUE,
                                     solidHeader = TRUE, status = "primary",
                                     verbatimTextOutput(ns("summaryIngredients")),  ## Check values
??
                                     radioButtons(ns("typeboxplotIngredient"), "Format type:",
```

```
                                                choices = c(excel = "xlsx", csv = "csv"), inline =
TRUE),
                                    downloadButton(ns("dwdboxplotIngredient"), "Download raw data",
                                                   style = "color: #fff; background-color: #27ae60;
border-color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                                    plotlyOutput(ns("boxplotIngredient"))
                                  )
                                )
                        ),
                        tabPanel("Generic stats",
                                 fluidRow(
                                   box(title = "Ingredients selection", solidHeader = TRUE, status =
"primary", width = 12,
                                       selectizeInput(
                                         inputId = ns("parentIngredientList"),
                                         label = "Ingredients :",
                                         multiple = FALSE,
                                         choices = NULL,
                                       )
                                   )
                                 ),
                                 fluidRow(
                                   # textOutput(ns("propProd")),
                                   # textOutput(ns("propPres")),
                                   shinydashboard::valueBoxOutput(ns("ParentpropProd"), width = 3),
                                   shinydashboard::valueBoxOutput(ns("ParentpropPres"), width = 3),
                                   ## Statut :
                                   box(title = "Informations", width = 6,
                                       solidHeader = TRUE, status = "primary", collapsible = TRUE,
                                       htmlOutput(ns("Parentfonction")),
                                       htmlOutput(ns("Parenttoxicity")),
                                       textOutput(ns("ParentonMarket"))
                                   ),

                                 ),

                                 fluidRow(
                                   box(title = "Product which contain the selected ingredient",
                                       solidHeader = TRUE, status = "primary", collapsible = TRUE,
                                       radioButtons(ns("typeParentproductOnMarket"), "Format type:",
                                                    choices = c(excel = "xlsx", csv = "csv"), inline =
TRUE),

                                       downloadButton(ns("dwdParentproductOnMarket"), "Download
table",
                                                      style = "color: #fff; background-color: #27ae60;
border-color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                                       DTOutput(ns("ParentproductOnMarket"))
                                   ),
                                   box(title = "Summary of ingredient", collapsible = TRUE,
                                       solidHeader = TRUE, status = "primary",
                                       verbatimTextOutput(ns("ParentsummaryIngredients")),  ## Check
values ??

                                       radioButtons(ns("typeParentboxplotIngredient"), "Format type:",
                                                    choices = c(excel = "xlsx", csv = "csv"), inline =
TRUE),
                                       downloadButton(ns("dwdParentboxplotIngredient"), "Download raw
data",
                                                      style = "color: #fff; background-color: #27ae60;
border-color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                                       plotlyOutput(ns("ParentboxplotIngredient"))
                                   )
                                 )
                        )
                 ),
    )
}

#' ingredients Server Functions
#'
#' @importFrom stringi stri_detect_fixed
#' @importFrom stats ave
#' @importFrom utils write.csv
#' @importFrom data.table setDT
#'
#' @noRd
mod_ingredients_server <- function(id, r){
  moduleServer( id, function(input, output, session){
```

```
    ns <- session$ns

    observe({

      req(r$dataBase)

      t1 <- Sys.time()

      ## get data :
      sub <- r$dataBase$LastEcigSubmission
      ingredients <- r$dataBase$LastEcigIngredient
      recip <- r$dataBase$LastEcigRecipeSubst
      map <- r$dataBase$MapIngSubst
      mapFunct <- r$dataBase$MapEcigIngCountFunction
      product <- r$dataBase$LastEcigProduct
      presentation <- r$dataBase$LastEcigPresentation
      # submitters <- r$dataBase$LastEcigSubmission
      recip <- r$dataBase$LastEcigRecipeSubst
      RefIngredientFunction <- r$dataBase$RefIngredientFunction
      RefToxicityStatus <- r$dataBase$RefToxicityStatus

      # Format recip :
      data.table::setDT(recip) # create index to make if faster
      data.table::setDT(mapFunct) # create index to make if faster
      recip <- merge(recip, mapFunct, by = c("SubmissionID", "IngredientCount")) # add ingredient
function
      recip <-  merge(recip, ingredients[, c("SubmissionID", "IngredientCount", "ToxicityStatus")],
by = c("SubmissionID", "IngredientCount")) # add ingredient tox status

      recip[is.na((recip$GenericSubstName)),]$SubstID <- "IN_PROGRESS"
      recip[is.na((recip$GenericSubstName)),]$SubstCAS <- "IN_PROGRESS"
      recip[is.na((recip$GenericSubstName)),]$SubstName <- "IN_PROGRESS"
      recip[is.na((recip$GenericSubstName)),]$GenericSubstName <- "IN_PROGRESS"
      recip$RecipeQuantity[recip$RecipeQuantity < 0] <- 0

      # Dataframe to get the list of ingredients
      df <- as.data.frame(table(recip$SubstCAS))
      colnames(df) <- c("CASNumber", "Frequency")
      df <- df %>% arrange(desc(df$Frequency))
      df$name <- recip[match(x = df$CASNumber, table = recip$SubstCAS ),]$SubstName
      df$ParentSubstName <- recip[match(x = df$CASNumber, table = recip$SubstCAS
),]$GenericSubstName
      ####

      ##############################################################################################

      updateSelectizeInput(session = session, inputId = "ingredientsList", choices =
unique(paste(df$CASNumber, df$name, sep = " ")), server = TRUE)
      updateSelectizeInput(session = session, inputId = "parentIngredientList", choices =
unique(recip$GenericSubstName), server = TRUE)
      ### New to check
      observeEvent(input$ingredientsList, {
        print(input$ingredientsList)
        cas <- strsplit(x = input$ingredientsList, split = " ")[[1]][1] # get the cas of the
ingredient
        # print(cas)

        # Use of CAS to compute the number of product who has the ingredient :
        proportionProduct <- (length(unique(recip[recip$SubstCAS == cas, ]$ProductID)) /
                                length(unique(recip$ProductID))) * 100


        proportionPresentation <- (dim(presentation[presentation$ProductID %in%
unique(recip[recip$SubstCAS == cas, ]$ProductID), ])[[1]] /
                                     dim(presentation)[[1]]) * 100

        output$propProd <- shinydashboard::renderInfoBox({
          valueBox(
            value = paste0(round(x = proportionProduct, digits = 3), " %"),
            subtitle = "products",
            width = 2,
            color = "blue",
            icon = icon("cube")
          )
        })
        output$propPres <- shinydashboard::renderInfoBox({
          valueBox(
            value = paste0(round(x = proportionPresentation, digits = 3), " %"),
```

```
            subtitle = "presentation",
            width = 2,
            color = "light-blue",
            icon = icon("cubes")
          )
        })

        # Get the function of the substance :
        funct <- names(which.max(table(recip[recip$SubstCAS == cas, ]$Function)))
        output$fonction <- renderText({
          paste0("The function of the selected product is mostly classified as <b>",
                 RefIngredientFunction[RefIngredientFunction$item == funct, ]$label_en, "</b>")
        })

        # Get the toxicity status of the substance :
        tox <- names(which.max(table(recip[recip$SubstCAS == cas, ]$ToxicityStatus)))
        output$toxicity <- renderText({
          paste0("Most declared toxicity : <b>",
                 RefToxicityStatus[RefToxicityStatus$item == tox, ]$label_en, "</b>")
        })

        # a verifier si OK ?
        onMarket <- NULL
        if("active" %in% sub[unique(recip[recip$SubstCAS == cas, ]$ProductID) %in%
sub$ProductID,]$List){
          output$onMarket <- renderText({
            paste0("Ingredient present in products on the market")
          })

          ##### Product repartition :
          output$dwdproductOnMarket <- downloadHandler(
            filename = function() {
              paste("ProductOnMarket", Sys.Date(), input$typeproductOnMarket, sep=".")
            },
            content = function(file) {
              if(input$typeproductOnMarket == "xlsx") {
                writexl::write_xlsx(sub[sub$ProductID %in% unique(recip[recip$SubstCAS == cas,
]$ProductID),
                                        c("SubmitterID", "ProductID", "LastUpdate",
"List")][input$productOnMarket_rows_all,], file)
              } else {
                write.csv(sub[sub$ProductID %in% unique(recip[recip$SubstCAS == cas, ]$ProductID),
                              c("SubmitterID", "ProductID", "LastUpdate",
"List")][input$productOnMarket_rows_all,],
                          file= file,
                          row.names=F)
              }
            }
          )

          output$productOnMarket <- renderDT({
            datatable(
              data = sub[sub$ProductID %in% unique(recip[recip$SubstCAS == cas, ]$ProductID),
                         c("SubmitterID", "ProductID", "LastUpdate", "List")],
              rownames = FALSE,
              filter = 'top',
              selection = "none",
              options = list(
                dom = 'Bfrtip',
                deferRender = TRUE,
                scrollY = 400,
                scroller = TRUE
              )
            )

          })


        } else {
          output$onMarket <- renderText({
            paste0("Ingredient not in products on the market")
          })
        }

        ## Box plot for summary of the selected ingredients
        output$summaryIngredients <- renderPrint({
          summary(recip[recip$SubstCAS == cas, ]$RecipeConcPPM)
        })
```

```
        ###### Ingrediient data dwd:
        output$dwdboxplotIngredient <- downloadHandler(
          filename = function() {
            paste("ingredientSummary", Sys.Date(), input$typeboxplotIngredient, sep=".")
          },
          content = function(file) {
            if(input$typeboxplotIngredient == "xlsx") {
              writexl::write_xlsx(recip[recip$SubstCAS == cas, ], file)
            } else {
              write.csv(recip[recip$SubstCAS == cas, ],
                        file= file,
                        row.names=F)
            }
          }
        )
        ## Graphical representation of the summary information
        output$boxplotIngredient <- renderPlotly({
          plot_ly(data = recip[recip$SubstCAS == cas, ], y= ~RecipeConcPPM, type = "box", name = "",
hoverinfo = "y") %>%
            layout(title = paste0("Boxplot of ", unique(recip[recip$SubstCAS == cas, ]$SubstCAS)),
                   yaxis = list(type = "log"), xaxis = list(visible = FALSE))
        })
      })


      ### For the second onglet parents
      observeEvent(input$parentIngredientList, {

        req(input$parentIngredientList)

        cas <- strsplit(x = input$parentIngredientList, split = " ")[[1]][1] # get the cas of the
ingredient

        # Use of CAS to compute the number of product who has the ingredient
        proportionProduct <- (length(unique(recip[recip$GenericSubstName ==
input$parentIngredientList, ]$ProductID)) /
                                length(unique(ingredients$ProductID))) * 100

        proportionPresentation <- (dim(presentation[presentation$ProductID %in%
unique(recip[recip$GenericSubstName == input$parentIngredientList, ]$ProductID), ])[[1]] /
                                     dim(presentation)[[1]]) * 100

        # Get the function of the substance :
        funct <- names(which.max(table(recip[recip$GenericSubstName == input$parentIngredientList,
]$Function)))

        output$Parentfonction <- renderText({
          paste0("The function of the selected product is mostly classified as <b>",
                 RefIngredientFunction[RefIngredientFunction$item == funct, ]$label_en, "</b>")
        })

        # Get the toxicity status of the substance :
        tox <- names(which.max(table(recip[recip$GenericSubstName == input$parentIngredientList,
]$ToxicityStatus)))
        output$Parenttoxicity <- renderText({
          paste0("Most declared toxicity : <b>",
                 RefToxicityStatus[RefToxicityStatus$item == tox, ]$label_en, "</b>")
        })

        output$ParentpropProd <- shinydashboard::renderInfoBox({
          valueBox(
            value = paste0(round(x = proportionProduct, digits = 3), " %"),
            subtitle = "products",
            width = 2,
            color = "blue",
            icon = icon("cube")
          )
        })
        output$ParentpropPres <- shinydashboard::renderInfoBox({
          valueBox(
            value = paste0(round(x = proportionPresentation, digits = 3), " %"),
            subtitle = "presentation",
            width = 2,
            color = "light-blue",
            icon = icon("cubes")
          )
```

```
        })

        # Check if the ingredient is in a product currently on the market.
        onMarket <- NULL
        if("active" %in% sub[unique(recip[recip$GenericSubstName == input$parentIngredientList,
]$ProductID) %in% sub$ProductID,]$List){
          output$ParentonMarket <- renderText({
            paste0("Ingredient present in products on the market")
          })

          ###### Ingredient data dwd:
          output$dwdParentproductOnMarket <- downloadHandler(
            filename = function() {
              paste("ParentproductOnMarket", Sys.Date(), input$typeParentproductOnMarket, sep=".")
            },
            content = function(file) {
              if(input$typeParentproductOnMarket == "xlsx") {
                writexl::write_xlsx(sub[sub$ProductID %in% unique(recip[recip$GenericSubstName ==
input$parentIngredientList, ]$ProductID),
                                        c("SubmitterID", "ProductID", "LastUpdate",
"List")][input$ParentproductOnMarket_rows_all,], file)
              } else {
                write.csv(latest[latest$ProductID %in%
unique(ingredients[ingredients$ParentSubstName == input$parentIngredientList, ]$ProductID),
                                    c("SubmitterID", "ProductID", "SubmissionDate",
"List")][input$ParentproductOnMarket_rows_all,],
                          file= file,
                          row.names=F)
              }
            }
          )

          output$ParentproductOnMarket <- renderDT({
            datatable(
              data = sub[sub$ProductID %in% unique(recip[recip$GenericSubstName ==
input$parentIngredientList, ]$ProductID),
                         c("SubmitterID", "ProductID", "LastUpdate", "List")],
              rownames = FALSE,
              filter = 'top',selection = "none",
              options = list(
                dom = 'Bfrtip',
                deferRender = TRUE,
                scrollY = 400,
                scroller = TRUE
              )
            )

          })
        } else {
          output$ParentonMarket <- renderText({
            paste0("Ingredient not in products on the market")
          })
        }
        #
        #   ## Add frequences to ingredients for each products:
        #   tmp <- transform(ingredients,                  # Calculate percentage by group
        #                    perc = ave(RecipeQuantity,
        #                               ProductID,
        #                               FUN = prop.table))
        #   tmp$ppm <- tmp$perc * 10^6
        ## Box plot for summary of the selected ingredients
        output$ParentsummaryIngredients <- renderPrint({
          summary(recip[recip$GenericSubstName == input$parentIngredientList, ]$RecipeConcPPM)
        })

#########################################################################################################
#############
        output$dwdParentproductOnMarket <- downloadHandler(
          filename = function() {
            paste("ParentproductOnMarket", Sys.Date(), input$typeParentproductOnMarket, sep=".")
          },
          content = function(file) {
            if(input$typeParentproductOnMarket == "xlsx") {
              writexl::write_xlsx(recip[recip$GenericSubstName == input$parentIngredientList, ],
file)
            } else {
              write.csv(tmp[recip$GenericSubstName == input$parentIngredientList, ],
                        file= file,
```

```
                        row.names=F)
            }
          }
        )

        # Download the data of the boxplot
        output$dwdParentboxplotIngredient <- downloadHandler(
          filename = function() {
            paste("ingredientSummary", Sys.Date(), input$typeParentboxplotIngredient, sep=".")
          },
          content = function(file) {
            if(input$typeParentboxplotIngredient == "xlsx") {
              writexl::write_xlsx(recip[recip$GenericSubstName == input$parentIngredientList, ],
file)
            } else {
              write.csv(recip[recip$GenericSubstName == input$parentIngredientList, ],
                        file= file,
                        row.names=F)
            }
          }
        )

        ## Graphical representation of the summary information
        output$ParentboxplotIngredient <- renderPlotly({
          plot_ly(data = recip[recip$GenericSubstName == input$parentIngredientList, ], y=
~RecipeConcPPM, type = "box", name = "", hoverinfo = "y") %>%
            layout(title = paste0("Boxplot of ", unique(recip[recip$GenericSubstName ==
input$parentIngredientList, ]$SubstCAS)),
                   yaxis = list(type = "log"), xaxis = list(visible = FALSE))
        })
      })

      t2 <- Sys.time()
      print("Temps ing : ")
      print(t2 - t1)

    })
  })
}

## To be copied in the UI
# mod_ingredients_ui("ingredients_1")

## To be copied in the server
# mod_ingredients_server("ingredients_1")
```

7. Source code: <EC> mod_inputData.R

```
# mod_inputData.R version 23-01-2024
#
# Copyright (c) Thomas El Khilali - euceg@anses.fr 2022-2023 (JATC2-WP5)
# Except specific files which bear a different mention, this programme is licensed under the EUPL-
1.2 or later
# You may obtain a copy of the license at https://joinup.ec.europa.eu/collection/eupl/eupl-text-
eupl-12
#
# This activity has received funding from the European Union's Health Program (2014-2020) under
grant agreement N°101035968 (JA-01-2020 - HP-JA-2020 - HP-JA-2020-2).
# The content of this document represents the views of the author only and is his/her sole
responsibility; it cannot be considered to reflect the views of the European Commission and/or the
European Health and Digital Executive
# Agency (HaDEA) or any other body of the European Union. The European Commission and the Agency do
not accept any responsibility for use that may be made of the information it contains.


#' inputData UI Function
#'
#' @description Module to import the database into the app
#'
#' @param id,input,output,session Internal parameters for {shiny}.
#'
#' @importFrom shinyFiles shinyFilesButton getVolumes shinyFileChoose
#' @importFrom DT DTOutput
#' @import shinydashboard
#' @importFrom utils write.csv
#' @importFrom shinyBS bsTooltip
#'
#' @noRd
#'
#' @importFrom shiny NS tagList
mod_inputData_ui <- function(id){
  ns <- NS(id)

  tabItem(tabName = "data",
          tags$head(tags$style(".shiny-notification {position: fixed; top: 60% ;left: 50%")),
          tabBox(id = "t1", width = 12,
                 # tabPanel(title = "About", icon=icon("address-card"),
                 #          h2("About this tool"),
                 #          tags$p("This app aims to provide a tool to analyse and describe
                 #                 data from producer declaration available on EU-CEG.",br(),
                 #                 "Sed mauris nisi, sollicitudin nec rutrum a, vestibulum vitae
orci.
                 #                 Mauris at suscipit tellus. Pellentesque habitant morbi tristique
senectus
                 #                 et netus et malesuada fames ac turpis egestas. In vehicula est
                 #                 lectus, eget tincidunt purus sodales non. Ut ultricies, velit
                 #                 sollicitudin volutpat porttitor, ligula neque posuere nibh,
                 #                 ornare ullamcorper urna eros quis erat. Ut hendrerit mauris nec
                 #                 elementum congue. Praesent in ex nisi."
                 #          ),
                 #          h2("Contact us"),
                 #          tags$p("If you have troubles using the app or if you have any
                 #                  questions or feedback relating to the data or the tool, then
                 #                 please contact us at xxxxxxx@anses.fr and we will be happy to
help.")
                 # ),
                 tabPanel(title = "Data",icon = icon("table"),
                          shinyFiles::shinyFilesButton(ns("Btn_GetFile"), "Choose a file" ,
                                                       title = "Please select a file:", multiple =
FALSE,
                                                       buttonType = "default", class = NULL),
                          # actionButton(ns("get"), "Choose the database"),

                          tags$i(
                             class = "glyphicon glyphicon-info-sign",
                             style = "color:#0072B2;",
                             title = "Please, choose the database file format .db"
                          ),


                          shinyBS::bsTooltip(id = ns("Btn_GetFile"), title = "Please, choose the
database file format .db",
```

```
                                placement = "left", trigger = "hover"),
                        tags$div(uiOutput(ns("formatDownload")),
                                 style= "display:inline-block; margin-right:2.5%; margin-
left:2.5%"),
                        tags$div(uiOutput(ns("buttonDownload")),
                                 style ="display:inline-block; margin-right:2.5%; margin-
left:2.5%"),
                        uiOutput(ns("listTables")),
                        DT::DTOutput(ns("dataFrame"))

                )
        )
  )
}

#' inputData Server Functions
#'
#' @import DBI
#' @import RSQLite
#' @importFrom DT renderDT datatable
#' @importFrom writexl write_xlsx
#' @importFrom shinyBS addTooltip
#' @importFrom shinyFiles getVolumes shinyFileChoose parseFilePaths
#'
#' @noRd
mod_inputData_server <- function(id, r){
  moduleServer(id, function(input, output, session){
    ns <- session$ns

    t1 <- Sys.time() # test de temps

    volumes = shinyFiles::getVolumes()

    dataBase <- list()

    # val = reactiveVal()

    # observeEvent(input$get, {
    #   val(file.choose())
    # })

      observe({
        # req(val())
        # shinyjs::toggle(id = "type", anim = FALSE, animType = "slide", time = 0.5, selector =
NULL, condition = r$dataBase)
        shinyFiles::shinyFileChoose(input = input,id = "Btn_GetFile",
                                    roots = volumes, session = session, filetypes= "db")

        file_selected <- shinyFiles::parseFilePaths(roots = volumes,
                                                    selection =  req(input$Btn_GetFile))

        # output$txt_file <- renderText(as.character(req(file_selected$datapath)))

        # Open the link to the SQL data base
        db <- DBI::dbConnect(RSQLite::SQLite(), as.character(req(file_selected$datapath)))

        # db <- DBI::dbConnect(RSQLite::SQLite(), as.character(val()))

        # tmp filter to speed up tests : less table = less loading time
        filter <- c("EcigAttachment", "EcigEmission","EcigEntityDetails",
"EcigIngNotMap","EcigIngredient","EcigPresentation",
                    "EcigProduct", "EcigRecipe",
"EcigRecipeStats","EcigRecipeSubst","EcigSalesData",
                    "EcigSubmitterHierarchy", "LastEcigAttachment", "LastEcigEmission",
                    "LastEcigEntityDetails",
                    "LastEcigSubmitterHierarchy", "LastSubmitterDetails",
                    "RefEcigEmissionName",
                    "RefEcigVoltageWattageAdjustable", "RefSubmissionType",
                    "RefToxicologicalDataAvailable",   "SubmitterDetails",
"EcigIngredientFunction")
        # keep :
"LastEcigSalesData","LastEcigPresentation","LastEcigProduct","LastEcigSubmitterDetails"
        # "RefEcigProductType", "LastEcigSubmission" , "LastEcigIngredient", "LastEcigRecipeSubst"
"MapIngSubst",
        # "MapEcigIngCountFunction" , "EcigSubmission", "LastEcigRecipSubst"

        # adapte names for the structur depending on the number of loaded tables
        names <- DBI::dbListTables(db)[!DBI::dbListTables(db) %in% filter]
```

```
        # Progress bar
        withProgress(message = 'Loading...', value = 0, {

          n <- length(DBI::dbListTables(db)) - length(filter) + 1 # number of time we go through the
loop

          # Go through tables :
          for(tbl in DBI::dbListTables(db)){
            # ... exept the one chosen in the filter
            if(! tbl %in% filter){
              incProgress(1/n,
                          detail = paste("Importing ", tbl)) # progress bar
              print(tbl) # dbg
              # Read the table
              df <- DBI::dbReadTable(conn = db, name = tbl)
              # Store it into the database structure
              dataBase[[length(dataBase)+1]]<- df
            } # end if
          } # end for

          names(dataBase) <- names # give names of each list component
        }) # end progress Bar

        DBI::dbDisconnect(conn = db) # disconnect the link to the database

        ### Visualisation :
        # Select the table to display
        output$listTables <- renderUI({
          selectInput(inputId = ns("tables"), label = "Select tables", choices =
names(dataBase[!startsWith(names(dataBase), "Ref")]))
        })


        ## To download data :
        output$dwd <- downloadHandler(
          filename = function() {
            paste(input$tables, Sys.Date(), input$type, sep=".")
          },
          content = function(file) {
            if(input$type == "xlsx") {
              writexl::write_xlsx(dataBase[[input$tables]][input$dataFrame_rows_all,], file)
            } else {
              write.csv(dataBase[[input$tables]][input$dataFrame_rows_all,],
                        file= file,
                        row.names=F)
            }
          }
        )

        # Display raw tables once they are loaded
        output$dataFrame <- DT::renderDT({

          req(input$tables)

          DT::datatable( ## to do : continue to add buttons to ease the use of dt (download
button...)
            data = as.data.frame(dataBase[input$tables]),
            filter = 'top',
            rownames = FALSE,
            fillContainer = TRUE,
            class = "compact hover stripe",
            selection = "none",
            options = list(pageLength = 10, autoWidth = TRUE, scrollX=TRUE, scrollY = "400px",
scrollCollapse=TRUE)
          )

        })

        r$dataBase <- dataBase # store database for other modules (global scope throughr$dataBase)

        ## test success message after importation
        if(!is.null(r$dataBase)){
          output$formatDownload <- renderUI(
            radioButtons(ns("type"), "Format type:",
                         choices = c(excel = "xlsx", csv = "csv"), inline = TRUE)
          )
          output$buttonDownload <- renderUI(
```

```
            downloadButton(ns("dwd"), "Download table",
                           style = "color: #fff; background-color: #27ae60; border-color:
#fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;")
          )
          # req(input$tables)
          sendSweetAlert(
            session = session,
            title = "Data imported",
            text = "You may have to wait a minute before going further",
            type = "success"
          )
        }

    }) # end observe
      t2 <- Sys.time()
      print("Temps input data")
      print(t2-t1)
  }) # end module
} # end server

## To be copied in the UI
# mod_inputData_ui("inputData_1")

## To be copied in the server
# mod_inputData_server("inputData_1")
```

8. Source code: <EC> mod_presentation.R

```
# mod_presentation.R version 23-01-2024
#
# Copyright (c) Thomas El Khilali - euceg@anses.fr 2022-2023 (JATC2-WP5)
# Except specific files which bear a different mention, this programme is licensed under the EUPL-1.2 or later
# You may obtain a copy of the license at https://joinup.ec.europa.eu/collection/eupl/eupl-text-eupl-12
#
# This activity has received funding from the European Union's Health Program (2014-2020) under grant agreement N°101035968 (JA-01-2020 - HP-JA-2020 - HP-JA-2020-2).
# The content of this document represents the views of the author only and is his/her sole responsibility; it cannot be considered to reflect the views of the European Commission and/or the European Health and Digital Executive
# Agency (HaDEA) or any other body of the European Union. The European Commission and the Agency do not accept any responsibility for use that may be made of the information it contains.


#' presentation UI Function
#'
#' @description A shiny Module.
#'
#' @param id,input,output,session Internal parameters for {shiny}.
#'
#' @noRd
#'
#' @importFrom shiny NS tagList
mod_presentation_ui <- function(id){
  ns <- NS(id)
  tabItem(tabName = "presentations",
          fluidPage(
            # Icon colors :
            tags$style(".okicon {color:#008000}"),
            tags$style(".removeicon {color:#FF0000}"),
            tags$style(".hourglassicon {color:#FFA500}"),

            fluidRow(
              box(title = "Filters", solidHeader = TRUE, width = 8, status = "primary",
                  collapsible = TRUE,

                  column(width = 4,
                         selectizeInput(
                           inputId = ns("submitterId"),
                           label = "Submitter :",
                           multiple = FALSE,
                           choices = NULL
                         )
                  ),

                  column(width = 4,
                         selectizeInput(
                           inputId = ns("brandName"),
                           label = "Brand Name :",
                           multiple = FALSE,
                           # choices = paste(presentation$BrandName, presentation$BrandSubtypeName, sep = " "),
                           choices = NULL
                         )
                  )
              ),

              box(title = "Informations", solidHeader = TRUE, width = 4, collapsible = TRUE, status = "primary",
                  htmlOutput(ns("nbProducts")),
                  htmlOutput(ns("productType")),
                  htmlOutput(ns("nicotineConcentration"))
              )


            ),
            #UI FIGURES
            fluidRow(
              column(width = 2,
                     selectizeInput(
                       inputId = ns("relatedProducts"),
```

```
                      label = "Related products :",
                      multiple = FALSE,
                      choices = NULL
                    ),

              ),
              column(width = 10,
                     shinydashboard::valueBoxOutput(ns("firstSubmissionDateBox"), width = 3),
                     shinydashboard::valueBoxOutput(ns("lastestSubmissionDateBox"), width = 3)
              ),
            ),
            fluidRow(
              box(title = "Presentation related to the selected product", solidHeader = TRUE,
collapsible = TRUE, status = "primary",
                  width = 8,
                  tags$div(radioButtons(ns("typeRelatedProductsDf"), "Format type:",
                                        choices = c(excel = "xlsx", csv = "csv"), inline = TRUE),
                           style ="display:inline-block; margin-right:2.5%; margin-left:2.5%"),
                  tags$div(downloadButton(ns("dwdRelatedProductsDf"), "Download table",
                                          style = "color: #fff; background-color: #27ae60; border-
color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                           style ="display:inline-block; margin-right:2.5%; margin-left:2.5%"),
                  DTOutput(ns("relatedProductsDf"))
              ),
              box(title = "Icons", solidHeader = TRUE, collapsible = TRUE, status = "primary",
                  width = 4,
                  icon("ok", lib = "glyphicon", class = "okicon"),"The presentation is currently on
the market",br(),
                  icon("stop", lib = "glyphicon"),"The presentation was on the market but not
anymore",br(),
                  icon("remove", lib = "glyphicon", class = "removeicon", verify_fa = FALSE), "The
presentation never has been on the market",br(),
                  icon("hourglass", lib = "glyphicon", class = "hourglassicon"), "The presentation
is planned to be release on the market"
              )
            ),
            fluidRow(
              div(style = "background-color: #f2f2f2; height: 100%;",
                  div(style = "display: flex; align-items: center; justify-content: center; height:
100%;",
                      div(style = "background-color: #c4d4e8; border: 1px solid #ccc; padding: 20px;
max-width: 500px; text-align: center; border-radius: 10px; position: relative;",
                          p("To display sales data information and the list of ingredients, please
select a row in the dataframe above. It will select a presentation and show its informations"),
                          div(style = "font-size: 32px; color: black; background-color: #f2f2f2;
width: 50px; height: 50px; border-radius: 50%; display: flex; align-items: center; justify-content:
center; position: absolute; bottom: -25px; left: 50%; transform: translateX(-50%); border: 2px solid
black;",
                              HTML("&#8595;")
                          )
                      )
                  )
              ),
              tags$br(),
              br(),
              br()
            ),
            fluidRow(
              shinydashboard::valueBoxOutput(ns("launchDateBox"), width = 3),
              shinydashboard::valueBoxOutput(ns("withdrawalDate"), width = 3)
              # shinydashboard::valueBoxOutput(ns("nicotineConcentration"), width = 3)
            ),

            fluidRow(
              box(title = "Presentation Sales Data", solidHeader = TRUE, #width = 10,
                  collapsible = TRUE, status = "primary",
                  textOutput(ns("noSalesData")),
                  plotly::plotlyOutput(ns("prensetationSalesDataDF"))
              ),
              box(title = "Presentation Ingredients", solidHeader = TRUE,  #width = 10,
                  collapsible = TRUE, status = "primary",
                  tags$div(radioButtons(ns("typeIngredientsDf"), "Format type:",
                                        choices = c(excel = "xlsx", csv = "csv"), inline = TRUE),
                           style ="display:inline-block; margin-right:2.5%; margin-left:2.5%"),
                  tags$div(downloadButton(ns("dwdIngredientsDf"), "Download table",
                                          style = "color: #fff; background-color: #27ae60; border-
color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                           style ="display:inline-block; margin-right:2.5%; margin-left:2.5%"),
```

```
                DTOutput(ns("ingredientsDF"))
              )
            )
            # plotly::plotlyOutput(ns("salesDataProduct")),
          )
  )

  # tabItem(tabName = "presentations",
  #         tabsetPanel(type = "tabs",
  #                     tabPanel("Presentation browser",
  #                              fluidRow(
  #                                box(title = "Filters", solidHeader = TRUE, status = "primary",
width = 8,
  #                                    collapsible = TRUE,
  #                                    column(width = 4,
  #                                           selectizeInput(
  #                                             inputId = ns("submitterId"),
  #                                             label = "Submitter :",
  #                                             multiple = FALSE,
  #                                             choices = NULL,
  #                                           )
  #                                    ),
  #                                    column(width = 4,
  #                                           selectizeInput(
  #                                             inputId = ns("brandName"),
  #                                             label = "Brand Name :",
  #                                             multiple = FALSE,
  #                                             # choices = paste(presentation$BrandName,
presentation$BrandSubtypeName, sep = " "),
  #                                             choices = NULL
  #                                           )
  #                                    )
  #                                ),
  #
  #                                box(title = "Informations", solidHeader = TRUE, status =
"primary", width = 4, collapsible = TRUE,
  #                                    htmlOutput(ns("nbProducts")),
  #                                    htmlOutput(ns("productType")),
  #                                    htmlOutput(ns("nicotineConcentration"))
  #                                )
  #
  #
  #                              ),
  #
  #
  #
  #                              # Icon colors :
  #                              tags$style(".okicon {color:#008000}"),
  #                              tags$style(".removeicon {color:#FF0000}"),
  #                              tags$style(".hourglassicon {color:#FFA500}"),
  #
  #                              #UI FIGURES
  #                              fluidRow(
  #                                fluidRow(
  #                                  column(width = 3,
  #                                         selectizeInput(
  #                                           inputId = ns("relatedProducts"),
  #                                           label = "Related products :",
  #                                           multiple = FALSE,
  #                                           choices = NULL
  #                                         )
  #                                         # uiOutput(ns("relatedProducts")),
  #                                  ),
  #                                  column(width = 9,
  #
shinydashboard::valueBoxOutput(ns("firstSubmissionDateBox"), width = 3),
  #
shinydashboard::valueBoxOutput(ns("lastestSubmissionDateBox"), width = 3)
  #                                  ),
  #                                ),
  #                                fluidRow(
  #                                  box(title = "Presentation Sales Data", solidHeader = TRUE,
status = "primary", collapsible = TRUE,
  #                                      width = 12,
  #                                      radioButtons(ns("typeRelatedProductsDf"), "Format type:",
  #                                                   choices = c(excel = "xlsx", csv = "csv"),
inline = TRUE),
```

```
  #                                       downloadButton(ns("dwdRelatedProductsDf"), "Download
table"),
  #                                       DTOutput(ns("relatedProductsDf"))
  #                                   )
  #                                 ),
  #                                 fluidRow(
  #                                   shinydashboard::valueBoxOutput(ns("launchDateBox"), width = 3),
  #                                   shinydashboard::valueBoxOutput(ns("withdrawalDate"), width = 3)
  #                                   # shinydashboard::valueBoxOutput(ns("nicotineConcentration"),
width = 3)
  #                                 )
  #                               ),
  #                               # fluidRow(
  #                               #   # shinydashboard::valueBoxOutput(ns("nbProducts"), width = 3),
  #                               #   shinydashboard::valueBoxOutput(ns("launchDateBox"), width = 3),
  #                               #   shinydashboard::valueBoxOutput(ns("withdrawalDate"), width =
3),
  #                               #   shinydashboard::valueBoxOutput(ns("nicotineConcentration"),
width = 3),
  #                               # ),
  #                               fluidRow(
  #                                 box(title = "Presentation Sales Data", solidHeader = TRUE, status
= "primary", #width = 10,
  #                                     collapsible = TRUE,
  #                                     textOutput(ns("noSalesData")),
  #                                     plotly::plotlyOutput(ns("prensetationSalesDataDF"))
  #                                 ),
  #                                 box(title = "Presentation Ingredients", solidHeader = TRUE,
status = "primary", #width = 10,
  #                                     collapsible = TRUE,
  #                                     DTOutput(ns("ingredients"))
  #                                 )
  #                               ),
  #                               # plotly::plotlyOutput(ns("salesDataProduct")),
  #
  #                     ),
  #                     tabPanel("Advanced presentation browser",
  #                               # fluidRow(
  #                               #    shinycssloaders::withSpinner(DTOutput(ns("productDF")))
  #                               # ),
  #                               # fluidRow(
  #                               #
  #                               #   column(width = 3,
  #                               #          uiOutput(ns("relatedProductsBrowse")),
  #                               #          ),
  #                               #   column(width = 9,
  #                               #
shinydashboard::valueBoxOutput(ns("lastestSubmissionDateBoxBrowse"), width = 3),
  #                               #
shinydashboard::valueBoxOutput(ns("firstSubmissionBoxBrowse"), width = 3)
  #                               #          )
  #                               # ),
  #                               # fluidRow(
  #                               #   DTOutput(ns("relatedProductsDfBrowse"))
  #                               # ),
  #                               # fluidRow(
  #                               #   # shinydashboard::valueBoxOutput(ns("nbProductsBrowse"), width
= 3),
  #                               #   shinydashboard::valueBoxOutput(ns("launchDateBoxBrowse"), width
= 3),
  #                               #   shinydashboard::valueBoxOutput(ns("withdrawalDateBrowse"),
width = 3),
  #                               #
shinydashboard::valueBoxOutput(ns("nicotineConcentrationBrowse"), width = 3),
  #                               # ),
  #                               # fluidRow(
  #                               #   box(title = "Presentation Sales Data", solidHeader = TRUE,
status = "primary",
  #                               #       collapsible = TRUE, #width = 10,
  #                               #       textOutput(ns("noSalesDataBrowse")),
  #                               #       plotly::plotlyOutput(ns("prensetationSalesDataDFBrowse"))
  #                               #   ),
  #                               #   box(title = "Presentation Ingredients", solidHeader = TRUE,
status = "primary", #width = 10,
  #                               #       collapsible = TRUE,
  #                               #       DTOutput(ns("ingredientsBrowse"))
  #                               #   )
  #                               # )
```

```
  #                       )
  #         )
  # )
}

#' presentation Server Functions
#'
#' @noRd
mod_presentation_server <- function(id, r){
  moduleServer( id, function(input, output, session){
    ns <- session$ns

    observe({

      req(r$dataBase)

      t1 <- Sys.time()

      # Format submitters table :
      submitters <- r$dataBase$LastEcigSubmitterDetails
      # Cleaning
      submitters$Country <- stringr::str_replace_all(string = submitters$Country, pattern=" ",
repl="")
      # Mapping countries names with full name and region
      submitters$countriesNames <- countrycode::countrycode(sourcevar = submitters$Country, origin =
"iso2c", destination = "country.name")
      submitters$region <- countrycode::countrycode(sourcevar = submitters$Country, origin =
"iso2c", destination = "continent")
      submitters$submitterIdName <- paste(submitters$SubmitterID, submitters$Name, sep = " - ")

      # load submitter list from the server side (big list)
      updateSelectizeInput(session = session, inputId = 'submitterId', choices = c("None",
submitters$submitterIdName), selected = "None", server = TRUE)
      # updateSelectizeInput(session = session, inputId = 'productType', choices = c("All",
unique(r$dataBase$RefEcigProductType$label)), selected = "All", server = TRUE)

      # Get sales :
      sales <- r$dataBase$LastEcigSalesData

      # Get product
      # first <- first[first$SubmitterID %in% submitters$SubmitterID, ]
      product <- r$dataBase$LastEcigProduct
      # Ingredients :
      ingredients <- r$dataBase$LastEcigRecipeSubst

      # Format brand Names and display them in drop list :
      presentation <- r$dataBase$LastEcigPresentation
      presentation$combinedBrandNames <- paste(presentation$BrandName,
presentation$BrandSubtypeName, sep = " ")
      presentation$combinedBrandNames <- gsub(pattern = " NA", replacement = "", x =
presentation$combinedBrandNames) # remove NA to clean names
      presentation$ProductType <- product$ProductType[match(presentation$ProductID,
product$ProductID)]
      presentation$ProductTypeName <-
r$dataBase$RefEcigProductType$label_en[match(presentation$ProductType,
r$dataBase$RefEcigProductType$item)]
      presentation$status <- r$dataBase$LastEcigSubmission$List[match(presentation$ProductID,
r$dataBase$LastEcigSubmission$ProductID)]

      updateSelectizeInput(session = session, inputId = 'brandName', choices = c("None",
presentation$combinedBrandNames), selected = "None", server = TRUE)

      df <- data.frame()

      observeEvent(input$submitterId,{
        req(input$submitterId)
        # If a submitter is selected : change the type and brand name
        if(input$submitterId == "None") {
          updateSelectizeInput(session = session, inputId = 'productType', choices = c("All",
unique(r$dataBase$RefEcigProductType$label)), selected = "All", server = TRUE)
          updateSelectizeInput(session = session, inputId = 'brandName', choices = c("None",
presentation$combinedBrandNames), selected = "None", server = TRUE)
        } else {
          id <- submitters[submitters$submitterIdName == input$submitterId,]$SubmitterID

          tp <- as.data.frame(table(product[product$SubmitterID %in% id,]$ProductType))
          colnames(tp) <- c("Type", "Quantity")
```

```
          tp$Type <- r$dataBase$RefEcigProductType[r$dataBase$RefEcigProductType$item %in%
tp$Type,]$label

          updateSelectizeInput(session = session, inputId = 'productType', choices = c("All",
unique(tp$Type)), selected = "All", server = TRUE)

          # Update brandnames :
          brandNms <- presentation[presentation$SubmitterID == id, ]$combinedBrandNames
          updateSelectizeInput(session = session, inputId = 'brandName', choices = c("None",
unique(brandNms)), selected = "None", server = TRUE)

        }

      })

      # Get product type to filter brand names :
      # observeEvent(input$productType,{
      #   req(input$productType)
      #   # print(input$productType)
      #
      #   if(input$productType != "All"){
      #     # Get the type
      #     nbType <- r$dataBaseRefEcigProductType[r$dataBaseRefEcigProductType$label %in%
input$productType, ]$item
      #     # Get the subID from the nb ID
      #     submissionID <- product[product$ProductType == nbType, ]$SubmissionID
      #     # Filter Presentation on submission ID :
      #     brandNms <- presentation[presentation$SubmissionID %in% submissionID, ]$BrandSubtypeName
      #     updateSelectizeInput(session = session, inputId = 'brandName', choices = c("None",
unique(brandNms)), selected = "None", server = TRUE)
      #     print("PRODUCT SELECTED")
      #     print(length(brandNms))
      #   } else if(input$productType == "All" && input$submitterId != "None"){
      #     id <- submitters[submitters$submitterIdName == input$submitterId,]$SubmitterID
      #     brandNms <- presentation[presentation$SubmitterID == id, ]$combinedBrandNames
      #     updateSelectizeInput(session = session, inputId = 'brandName', choices = c("None",
unique(brandNms)), selected = "None", server = TRUE)
      #     print("PRODUCT NON SELECTED")
      #     print(length(brandNms))
      #   } else if(input$productType == "All" && input$submitterId == "None"){
      #     updateSelectizeInput(session = session, inputId = 'brandName', choices = c("None",
presentation$combinedBrandNames), selected = "None", server = TRUE)
      #   }
      #
      # })

      ## Pour la suite once it is filtered :
      observeEvent(input$brandName, {
        req(input$brandName)

        ## Reset ingredients DT when brandName change.

        if(input$brandName != "None") {

          # Update related products based on
          # output$relatedProducts <- renderUI({
          #   selectizeInput(inputId = ns("relatedProducts"), label = "Related products :", multiple
= FALSE, choices = unique(presentation[presentation$combinedBrandNames == input$brandName,
]$ProductID))
          # })
          updateSelectizeInput(session = session, inputId = "relatedProducts", choices =
unique(presentation[presentation$combinedBrandNames == input$brandName, ]$ProductID), server = TRUE)

          ## Print Nb of related product (unique) for information
          # output$nbProducts <- shinydashboard::renderInfoBox({
          #   valueBox(
          #     value = length(unique(presentation[presentation$combinedBrandNames ==
input$brandName, ]$ProductID)),
          #     subtitle = "Number of related product(s)",
          #     width = 2,
          #     color = "purple",
          #     icon = icon("cube")
          #   )
          # })
          output$nbProducts <- renderText({

            paste("This brandname presentation was found in <b>",
```

```
                length(unique(presentation[presentation$combinedBrandNames == input$brandName,
]$ProductID)),
                " product(s) </b> <I>(check the droplist Related Product below)</I>")
          })

          output$productType <- renderText({
            paste("The selected presentation is declared as <b>",
                  unique(presentation[presentation$combinedBrandNames == input$brandName
                                      & presentation$ProductID == input$relatedProducts,
]$ProductTypeName,"</b>")
            )
          })

          output$lastestSubmissionDateBox <- shinydashboard::renderInfoBox({
            # presentation <- presentation[presentation$ProductID == input$relatedProducts, ]
            # strsplit(x = , split = " ")[[1]][1]
            req(input$relatedProducts)
            valueBox(
              value = strsplit(x =
r$dataBase$LastEcigSubmission[r$dataBase$LastEcigSubmission$ProductID == input$relatedProducts,
]$LastUpdate,
                               split = " ")[[1]][1], # Get only the date wich is the first element
because the string is too long
              subtitle = "Latest Submission",
              width = 2,
              color = "teal",
              icon = icon("arrows-rotate")
            )
          })

          output$firstSubmissionDateBox <- shinydashboard::renderInfoBox({
            req(input$relatedProducts)
            valueBox(
              value = strsplit(x =
r$dataBase$LastEcigSubmission[r$dataBase$LastEcigSubmission$ProductID == input$relatedProducts,
]$FirstSubmission,
                               split = " ")[[1]][1], # Get only the date wich is the first element
because the string is too long
              subtitle = "First Submission",
              width = 2,
              color = "teal",
              icon = icon("export", lib = "glyphicon")
            )
          })


          # output$nicotineConcentration <- shinydashboard::renderInfoBox({
          #   # presentation <- presentation[presentation$ProductID == input$relatedProducts, ]
          #   valueBox(
          #     value = paste(unique(product[product$ProductID == input$relatedProducts,
]$NicotineConcentration), " mg/mL"),
          #     subtitle = "Nicotine concentration",
          #     width = 2,
          #     color = "maroon",
          #     # icon = icon("remove", verify_fa = FALSE)
          #   )
          # })

          output$nicotineConcentration <- renderText({
            paste("Nicotine concentration : <b>",
                  unique(product[product$ProductID == input$relatedProducts,
]$NicotineConcentration),
                  " mg/mL</b>")
          })

          output$relatedProductsDf <- renderDT({
            req(input$relatedProducts)
            df <<- presentation[presentation$ProductID == input$relatedProducts, ]

            df <<- dplyr::select(df, c("PresentationCount", "combinedBrandNames", "ProductTypeName",
"LaunchDate", "WithdrawalDate", "status", "ProductID"))
            df$LaunchDate <<- as.Date(df$LaunchDate)
            df$WithdrawalDate <<- as.Date(df$WithdrawalDate)

            ## Check all dates to see if the presentation is available or not
            df$state <<- apply(X = df, MARGIN = 1, function(x) {
              # For each line of the DT, store values
              launch <- x[4] # get launchdate in df
```

```
              withdraw <- x[5] # get withdrawal date in df
              status <- x[6] # get status date in df

              if(!is.na(withdraw)){
                if((withdraw < launch) || (launch == withdraw && withdraw < as.Date(Sys.time()))) {
                  ico <- as.character(icon("remove", lib = "glyphicon", class = "removeicon",
verify_fa = FALSE)) #"Presentation removed from the market"
                } else if ((withdraw > launch && withdraw < as.Date(Sys.time()))) {
                  ico <- as.character(icon("stop", lib = "glyphicon")) #"Presentation removed from
the market"
                }
                else if (withdraw > launch && withdraw > as.Date(Sys.time()) && launch <
as.Date(Sys.time())) {
                  ico <- as.character(icon("ok", lib = "glyphicon", class = "okicon"))#"On market"
                } else if((launch == withdraw && launch > as.Date(Sys.time())) ||
                          (withdraw > launch && launch > as.Date(Sys.time()))) {
                  ico <- as.character(icon("hourglass", lib = "glyphicon", class =
"hourglassicon"))#"On market"
                }
              } else { # OK
                if(launch < as.Date(Sys.time())){
                  # ico <- as.character(icon("ok", lib = "glyphicon", class = "okicon"))#"On market"
                  if(!status == "inactive"){
                    ico <- as.character(icon("ok", lib = "glyphicon", class = "okicon"))#"On market"
                  } else {
                    ico <- as.character(icon("stop", lib = "glyphicon"))
                  }
                } else {
                  ico <- as.character(icon("hourglass", lib = "glyphicon", class =
"hourglassicon"))#"On market"
                }
              }
              # #hourglass for sablier (waiting)
              return(ico)
            })

            DT::datatable(
              data = dplyr::select(df, c("PresentationCount", "combinedBrandNames", "LaunchDate",
"WithdrawalDate", "state")),#df, "ProductTypeName",
              rownames = FALSE,
              escape = FALSE,
              selection = list(mode = 'single')
            )
          })

          output$dwdRelatedProductsDf <- downloadHandler(
            filename = function() {
              paste("otherPresentation",Sys.Date(), input$typeRelatedProductsDf, sep=".")
            },
            content = function(file) {
              if(input$typeRelatedProductsDf == "xlsx") {
                writexl::write_xlsx(dplyr::select(df, c("PresentationCount", "combinedBrandNames",
"LaunchDate", "WithdrawalDate")), file)
              } else {
                write.csv(dplyr::select(df, c("PresentationCount", "combinedBrandNames",
"LaunchDate", "WithdrawalDate")),
                          file= file,
                          row.names=F)
              }
            }
          )

          output$salesDataProduct <- plotly::renderPlotly({

            req(input$relatedProducts)

            print("Selected product : ")
            print(input$relatedProducts)

            # Filter the dates
            sales <- dplyr::filter(sales, sales$Year>=2016)
            sales <- dplyr::filter(sales, sales$Year<=as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]))

            # Select the product
            sales <- sales[sales$ProductID == input$relatedProducts, c("Year", "SalesVolume")]

            if(nrow(sales) > 0){
```

```
              dfSales <- stats::aggregate(.~Year, data = sales, FUN = sum)
              fig <- plotly::plot_ly(dfSales, x = ~Year, y = ~SalesVolume, name = 'SalesVolume',
type = 'bar')
            }

          })

          observeEvent(input$relatedProductsDf_rows_selected,{
            req(input$relatedProductsDf_rows_selected)
            # output$noSalesData <- NULL

            row <- df[input$relatedProductsDf_rows_selected, ] # vector with values of the selected
line
            # print(row)
            # print("Selected stuff in row")
            # print(row$ProductID)
            # print(row$PresentationCount)
            output$launchDateBox <- shinydashboard::renderInfoBox({
              valueBox(
                value = row$LaunchDate,
                subtitle = "Launch date",
                width = 2,
                color = "green",
                icon = icon("calendar")
              )
            })

            output$withdrawalDate <- shinydashboard::renderInfoBox({
              req(input$relatedProducts)
              presentation <- presentation[presentation$ProductID == input$relatedProducts, ]
              valueBox(
                value = row$WithdrawalDate,
                subtitle = "Withdrawal date",
                width = 2,
                color = "red",
                icon = icon("remove", verify_fa = FALSE)
              )
            })

            selectedPresentationSales <- sales[which( sales$ProductID == row$ProductID &
                                                      sales$PresentationCount ==
row$PresentationCount), c("Year", "SalesVolume")]
            # print(dim(selectedPresentationSales))
            if(dim(selectedPresentationSales)[1] == 0){
              output$prensetationSalesDataDF <- NULL
              output$noSalesData <- renderText({
                print("No presentation sales data")
              })
            } else {
              if(sum(selectedPresentationSales$SalesVolume) == 0){
                output$prensetationSalesDataDF <- NULL
                output$noSalesData <- renderText({
                  print("Declared sales data is null")
                })
              } else { # if sales data is available
                # print("Sales data : ")
                # print(selectedPresentationSales)
                # print("########################################")
                output$noSalesData <- NULL
                output$prensetationSalesDataDF <- renderPlotly({
                  fig <- plotly::plot_ly(selectedPresentationSales, x = ~Year, y = ~SalesVolume,
name = 'SalesVolume', type = 'bar')
                  fig %>% layout(xaxis = list(range = c(2015, as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]))))
                })

              }

            }

            output$ingredientsDF <- renderDT({
              # ingredient <- tbl_list$EcigIngredient
              # ingTable <- ingredients[ingredients$ProductID == "00001-17-00175", c("SubstCAS",
"SubstName", "RecipeQuantity")]

              ingTable <- ingredients[ingredients$ProductID == row$ProductID, c("SubstCAS",
"SubstName", "RecipeQuantity")]
```

```
            datatable(
              ingTable[order(ingTable$RecipeQuantity, decreasing=TRUE), ],
              rownames = FALSE,
              selection = "none",
              filter = 'top'
            )

          })
          ## To download data :
          output$dwdIngredientsDf <- downloadHandler(
            filename = function() {
              paste("ingredientsList",Sys.Date(), input$typeIngredientsDf, sep=".")
            },
            content = function(file) {
              if(input$typeIngredientsDf == "xlsx") {
                writexl::write_xlsx(ingredients[ingredients$ProductID == row$ProductID,
c("SubstCAS", "SubstName", "RecipeQuantity")][input$ingredientsDF_rows_all,], file)
              } else {
                write.csv(ingredients[ingredients$ProductID == row$ProductID, c("SubstCAS",
"SubstName", "RecipeQuantity")][input$ingredientsDF_rows_all,],
                          file= file,
                          row.names=F)
              }
            }
          )

        })

      } else {
        output$lastestSubmissionDateBox <- shinydashboard::renderInfoBox({
          valueBox(
            value = "None",
            subtitle = "Latest Submission",
            width = 2,
            color = "teal",
            icon = icon("arrows-rotate")
          )
        })
        output$firstSubmissionDateBox <- shinydashboard::renderInfoBox({
          valueBox(
            value = "None",
            subtitle = "First Submission",
            width = 2,
            color = "teal",
            icon = icon("export", lib = "glyphicon")
          )
        })
        # output$nbProducts <- shinydashboard::renderInfoBox({
        #   valueBox(
        #     value = 0,
        #     subtitle = "Number of related product(s)",
        #     width = 2,
        #     color = "purple",
        #     icon = icon("cube")
        #   )
        # })
        output$nbProducts <- renderText({
          paste("This brandname presentation was found in ", 0,  " product(s)")

        })
        output$productType <- renderText({
          paste("The selected presentation is ...")
        })

        output$launchDateBox <- shinydashboard::renderInfoBox({
          valueBox(
            value = "None",
            subtitle = "Launch date",
            width = 2,
            color = "green",
            icon = icon("calendar")
          )
        })
        output$withdrawalDate <- shinydashboard::renderInfoBox({
          valueBox(
            value = "None",
            subtitle = "Withdrawal date",
            width = 2,
```

```
            color = "red",
            icon = icon("remove", verify_fa = FALSE)
          )
        })
        # output$nicotineConcentration <- shinydashboard::renderInfoBox({
        #   # presentation <- presentation[presentation$ProductID == input$relatedProducts, ]
        #   valueBox(
        #     value = paste(0, " mg/mL"),
        #     subtitle = "Nicotine concentration",
        #     width = 2,
        #     color = "maroon",
        #     # icon = icon("remove", verify_fa = FALSE)
        #   )
        # })
        output$nicotineConcentration <- renderText({
          paste("Nicotine concentration : <b>",
                0,
                " mg/mL</b>")
        })
        output$ingredientsDF <- NULL
        # output$prensetationSalesDataDF <- NULL
      }

    })

    ##### SECOND ONGLET : PRODUCT BROWSER :

    ## To decommente
    # entityAll <- r$dataBase$EcigEntityDetails[r$dataBase$EcigEntityDetails$SubmissionID %in%
latest$SubmissionID, ]
    # presentationAll <- r$dataBase$EcigPresentation[r$dataBase$EcigPresentation$SubmissionID %in%
latest$SubmissionID, ]
    # productAll <- r$dataBase$EcigProduct[r$dataBase$EcigProduct$SubmissionID %in%
latest$SubmissionID, ]
    # submissionAll <- r$dataBase$EcigSubmission[r$dataBase$EcigSubmission$SubmissionID %in%
latest$SubmissionID,]
    #
    # entityAll <- entityAll[, c("SubmissionID", "ProductID", "Name")]
    # presentationAll <- presentation[, c("SubmissionID", "BrandName", "BrandSubtypeName",
"combinedBrandNames", "NationalComment")]
    # productAll <- productAll[, c("SubmissionID", "Description")]
    # submissionAll <- submissionAll[, c("SubmissionID", "GeneralComment")]
    #
    # dfBrowser <- merge(x = entityAll, y =  presentationAll, by = "SubmissionID")
    # dfBrowser <- merge(x = dfBrowser, y =  productAll, by = "SubmissionID")
    # dfBrowser <- merge(x = dfBrowser, y =  submissionAll, by = "SubmissionID")


    output$productDF <- renderDT({
      datatable(
        dfBrowser,
        plugins = "ellipsis",
        filter = 'top',
        selection = list(mode = 'single'),
        rownames = FALSE,
        options = list(
          columnDefs = list(list(
            targets = c(1, 2, 3, 4, 5, 6, 7, 8),
            render = DT::JS("$.fn.dataTable.render.ellipsis( 50, false )")
          )))
      )
    })

    observeEvent(input$productDF_rows_selected,{
      req(input$productDF_rows_selected)

      row <- dfBrowser[input$productDF_rows_selected, ] # vector with values of the selected line
      # Contain :
      # "SubmissionID"       "ProductID"          "Name"               "BrandName"
"BrandSubtypeName"
      # "combinedBrandNames" "NationalComment"    "Description"        "GeneralComment"
      # print("ROOOOOWWWW")
      # print(colnames(row))

      ######################################### Change names :
      # Update related products based on
      output$relatedProductsBrowse <- renderUI({
        # print(dim(presentation))
```

```
          selectizeInput(inputId = ns("relatedProductsBrowse"), label = "Related products :",
multiple = FALSE, choices = unique(presentation[presentation$combinedBrandNames %in% row[6],
]$ProductID))
        })

        ########################################
        output$lastestSubmissionDateBoxBrowse <- shinydashboard::renderInfoBox({
          presentation <- presentation[presentation$ProductID %in% row[2], ]
          valueBox(
            value = min(unique(presentation[presentation$combinedBrandNames %in% row[6],
]$LaunchDate)),
            subtitle = "Latest Submission",
            width = 2,
            color = "teal",
            icon = icon("calendar")
          )
        })
        output$firstSubmissionBoxBrowse <- NULL

        ## Print Nb of related product (unique) for information
        # output$nbProductsBrowse <- shinydashboard::renderInfoBox({
        #   valueBox(
        #     value = length(unique(presentation[presentation$combinedBrandNames %in% row[6],
]$ProductID)),
        #     subtitle = "Number of related product(s)",
        #     width = 2,
        #     color = "purple",
        #     icon = icon("cube")
        #   )
        # })
        output$launchDateBoxBrowse <- shinydashboard::renderInfoBox({
          presentation <- presentation[presentation$ProductID %in% input$relatedProductsBrowse, ]
          valueBox(
            value = min(unique(presentation[presentation$combinedBrandNames %in% row[6],
]$LaunchDate)),
            subtitle = "Launch date",
            width = 2,
            color = "green",
            icon = icon("calendar")
          )
        })
        output$withdrawalDateBrowse <- shinydashboard::renderInfoBox({
          presentation <- presentation[presentation$ProductID %in% input$relatedProductsBrowse, ]
          valueBox(
            value = unique(presentation[presentation$combinedBrandNames %in% row[6],
]$WithdrawalDate),
            subtitle = "Withdrawal date",
            width = 2,
            color = "red",
            icon = icon("remove", verify_fa = FALSE)
          )
        })

        # output$nicotineConcentrationBrowse <- shinydashboard::renderInfoBox({
        #   valueBox(
        #     value = paste(unique(product[product$ProductID %in% row[2], ]$NicotineConcentration),
" mg/mL"),
        #     subtitle = "Nicotine concentration",
        #     width = 2,
        #     color = "maroon",
        #     # icon = icon("remove", verify_fa = FALSE)
        #   )
        # })

        output$nicotineConcentration <- renderText({
          paste("Nicotine concentration : <b>",
                unique(product[product$ProductID %in% row[2], ]$NicotineConcentration),
                " mg/mL</b>")
        })



        # output$relatedProductsDfBrowse <- renderDT({
        #   req(input$relatedProductsBrowse)
        #   df <<- presentation[presentation$ProductID == input$relatedProductsBrowse, ]
        #
        #   df <<- dplyr::select(df, c("PresentationCount", "combinedBrandNames", "ProductTypeName",
"LaunchDate", "WithdrawalDate", "status", "ProductID"))
```

```
        #   df$LaunchDate <<- as.Date(df$LaunchDate)
        #   df$WithdrawalDate <<- as.Date(df$WithdrawalDate)
        #
        #   ## Check all dates to see if the presentation is available or not
        #   df$state <<- apply(X = df, MARGIN = 1, function(x) {
        #     # For each line of the DT, store values
        #     launch <- x[4] # get launchdate in df
        #     withdraw <- x[5] # get withdrawal date in df
        #     status <- x[6] # get status date in df
        #
        #     if(!is.na(withdraw)){
        #       if((withdraw < launch) || (launch == withdraw && withdraw < as.Date(Sys.time()))) {
        #         ico <- as.character(icon("remove", lib = "glyphicon", class = "removeicon",
verify_fa = FALSE)) #"Presentation removed from the market"
        #       } else if ((withdraw > launch && withdraw < as.Date(Sys.time()))) {
        #         ico <- as.character(icon("stop", lib = "glyphicon")) #"Presentation removed from
the market"
        #       }
        #       else if (withdraw > launch && withdraw > as.Date(Sys.time()) && launch <
as.Date(Sys.time())) {
        #         ico <- as.character(icon("ok", lib = "glyphicon", class = "okicon"))#"On market"
        #       } else if((launch == withdraw && launch > as.Date(Sys.time())) ||
        #                 (withdraw > launch && launch > as.Date(Sys.time()))) {
        #         ico <- as.character(icon("hourglass", lib = "glyphicon", class =
"hourglassicon"))#"On market"
        #       }
        #     } else { # OK
        #       if(launch < as.Date(Sys.time())){
        #         # ico <- as.character(icon("ok", lib = "glyphicon", class = "okicon"))#"On market"
        #         if(!status == "inactive"){
        #           ico <- as.character(icon("ok", lib = "glyphicon", class = "okicon"))#"On market"
        #         } else {
        #           ico <- as.character(icon("stop", lib = "glyphicon"))
        #         }
        #       } else {
        #         ico <- as.character(icon("hourglass", lib = "glyphicon", class =
"hourglassicon"))#"On market"
        #       }
        #     }
        #     # #hourglass for sablier (waiting)
        #     return(ico)
        #   })
        #
        #   DT::datatable(
        #     data = dplyr::select(df, c("PresentationCount", "combinedBrandNames",
"ProductTypeName", "LaunchDate", "WithdrawalDate", "state")),#df,
        #     rownames = FALSE,
        #     escape = FALSE,
        #     selection = list(mode = 'single')
        #   )
        # })


        # observeEvent(input$relatedProductsDfBrowse_rows_selected,{
        #   req(input$relatedProductsDfBrowse_rows_selected)
        #   # output$noSalesData <- NULL
        #
        #   row <- df[input$relatedProductsDfBrowse_rows_selected, ] # vector with values of the
selected line
        #   # print(row)
        #
        #   selectedPresentationSales <- sales[which( sales$ProductID == row$ProductID &
        #                                               sales$PresentationCount ==
row$PresentationCount), c("Year", "SalesVolume")]
        #   # print(dim(selectedPresentationSales))
        #   if(dim(selectedPresentationSales)[1] == 0){
        #     output$prensetationSalesDataDF <- NULL
        #     output$noSalesDataBrowse <- renderText({
        #       print("No presentation sales data")
        #     })
        #   } else {
        #     if(sum(selectedPresentationSales$SalesVolume) == 0){
        #       output$prensetationSalesDataDF <- NULL
        #       output$noSalesDataBrowse <- renderText({
        #         print("Declared sales data is null")
        #       })
        #     } else { # if sales data is available
        #       # print("Sales data : ")
```

```
        #       # print(selectedPresentationSales)
        #       # print("########################################")
        #       output$prensetationSalesDataDFBrowse <- renderPlotly({
        #         fig <- plotly::plot_ly(selectedPresentationSales, x = ~Year, y = ~SalesVolume,
name = 'SalesVolume', type = 'bar')
        #       })
        #
        #     }
        #
        #   }
        #
        #   ## ADD INGREDIENT ONGLET :
        #   output$ingredientsBrowse <- renderDT({
        #     # ingredient <- tbl_list$EcigIngredient
        #     # ingredients[ingredients$ProductID == "00020-17-13114", c("CasClean", "UpName")]
        #     ingTable <- ingredients[ingredients$ProductID == row$ProductID, c("CasClean",
"UpName", "RecipeQuantity")]
        #     datatable(
        #       ingTable[order(ingTable$RecipeQuantity,decreasing=TRUE), ],
        #       rownames = FALSE
        #     )
        #
        #   })
        #
        # })



      })

      t2 <- Sys.time()
      print("Temps pres : ")
      print(t2 - t1)

    })
  })
}

## To be copied in the UI
# mod_presentation_ui("presentation_1")

## To be copied in the server
# mod_presentation_server("presentation_1")
```

9. Source code: <EC> mod_submitters.R

```
# mod_submitters.R version 23-01-2024
#
# Copyright (c) Thomas El Khilali - euceg@anses.fr 2022-2023 (JATC2-WP5)
# Except specific files which bear a different mention, this programme is licensed under the EUPL-
1.2 or later
# You may obtain a copy of the license at https://joinup.ec.europa.eu/collection/eupl/eupl-text-
eupl-12
#
# This activity has received funding from the European Union's Health Program (2014-2020) under
grant agreement N°101035968 (JA-01-2020 - HP-JA-2020 - HP-JA-2020-2).
# The content of this document represents the views of the author only and is his/her sole
responsibility; it cannot be considered to reflect the views of the European Commission and/or the
European Health and Digital Executive
# Agency (HaDEA) or any other body of the European Union. The European Commission and the Agency do
not accept any responsibility for use that may be made of the information it contains.


#' submitters UI Function
#'
#' @description A shiny Module.
#'
#' @param id,input,output,session Internal parameters for {shiny}.
#'
#' @import shinyWidgets
#' @import plotly
#' @import shinycssloaders
#' @importFrom shiny NS tagList
#'
#' @noRd
#'
mod_submitters_ui <- function(id){
  ns <- NS(id)
  tabItem(tabName = "submitters",
          fluidRow(
            box(title = "Filters", solidHeader = TRUE, status = "primary", width = 10,
                column(width = 3,
                       awesomeRadio(
                         inputId = ns("choice"),
                         label = "Submitter origin",
                         choices = c("All", "By Region","By Country"),
                         selected = "All",
                         checkbox = FALSE,
                         status = "primary"
                       ),
                ),
                column(width = 3,
                       uiOutput(ns("regionChoice")),
                       uiOutput(ns("countryChoice")),
                ),
                column(width = 4,
                       selectizeInput(
                         inputId = ns("submitterId"),
                         label = "Submitter :",
                         multiple = TRUE,
                         choices = NULL,
                         options = list(
                           placeholder = "Select a submitter by ID or name",
                           'plugins' = list('remove_button'),
                           'persist' = FALSE
                         )
                       )
                )
            )
          ),
          fluidRow(
            shinydashboard::valueBoxOutput(ns("boxSubmitters"), width = 4),
            shinydashboard::valueBoxOutput(ns("boxProduct"), width = 4),
            shinydashboard::valueBoxOutput(ns("boxPresentation"), width = 4)
          ),
          fluidRow(
            box(collapsible = TRUE, solidHeader = TRUE, title = "Product repartition", status =
"primary",
                tags$div(radioButtons(ns("type"), "Format type:",
                                      choices = c(excel = "xlsx", csv = "csv"), inline = TRUE),
```

```
                        style= "display:inline-block; margin-right:2.5%; margin-left:2.5%"
                ),
                tags$div(downloadButton(ns("dwdpieChart"), "Download raw data",
                                        style = "color: #fff; background-color: #27ae60; border-
color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                        style= "display:inline-block; margin-right:2.5%; margin-left:2.5%"
                ),
                shinycssloaders::withSpinner(plotly::plotlyOutput(outputId = ns("pieChart")))
            ),
            box(collapsible = TRUE, solidHeader = TRUE,title = "Brand Names", status = "primary",
                tags$div(radioButtons(ns("typeBrandname"), "Format type:",
                                      choices = c(excel = "xlsx", csv = "csv"), inline = TRUE),
                        style= "display:inline-block; margin-right:2.5%; margin-left:2.5%"
                ),
                tags$div(downloadButton(ns("dwdBrandname"), "Download table",
                                        style = "color: #fff; background-color: #27ae60; border-
color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                        style= "display:inline-block; margin-right:2.5%; margin-left:2.5%"
                ),
                DT::DTOutput(ns("brandname"))
            )
          ),
          fluidRow(
            box(collapsible = TRUE, solidHeader = TRUE, title = "Nicotine concentration per
product", status = "primary",
                tags$div(radioButtons(ns("typeNicotine"), "Format type:",
                                      choices = c(excel = "xlsx", csv = "csv"), inline = TRUE),
                        style= "display:inline-block; margin-right:2.5%; margin-left:2.5%"
                ),
                tags$div(downloadButton(ns("dwdNicotine"), "Download raw data",
                                        style = "color: #fff; background-color: #27ae60; border-
color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                        style= "display:inline-block; margin-right:2.5%; margin-left:2.5%"
                ),
                shinycssloaders::withSpinner(plotly::plotlyOutput(outputId = ns("nicotine")))
            ),
            box(collapsible = TRUE, solidHeader = TRUE, title = "Sales Data", status = "primary",
                tags$div(radioButtons(ns("typeSales"), "Format type:",
                                      choices = c(excel = "xlsx", csv = "csv"), inline = TRUE),
                        style= "display:inline-block; margin-right:2.5%; margin-left:2.5%"
                ),
                tags$div(downloadButton(ns("dwdSales"), "Download raw data",
                                        style = "color: #fff; background-color: #27ae60; border-
color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                        style= "display:inline-block; margin-right:2.5%; margin-left:2.5%"
                ),
                shinycssloaders::withSpinner(plotly::plotlyOutput(outputId = ns("salesData")))
            )
          )
    )
}

#' submitters Server Functions
#'
#' @importFrom countrycode countrycode
#' @importFrom stringr str_replace_all str_split
#' @import dplyr
#' @importFrom grDevices rainbow
#' @importFrom stats aggregate
#' @importFrom utils write.csv
#' @import pals
#'
#' @noRd
mod_submitters_server <- function(id, r){
  moduleServer( id, function(input, output, session){
    ns <- session$ns

    # Need observe to access r$dataBase
    observe({

      req(r$dataBase) # doesn't run if no data imported

      t1 <- Sys.time()

      #### Data preparation
      # Format submitters table :
      submitters <- r$dataBase$LastEcigSubmitterDetails
      # Cleaning
```

```
      submitters$Country <- stringr::str_replace_all(string=submitters$Country, pattern=" ",
repl="")
      # Mapping countries names with full name and region
      submitters$countriesNames <- countrycode::countrycode(sourcevar = submitters$Country, origin =
"iso2c", destination = "country.name")
      submitters$region <- countrycode::countrycode(sourcevar = submitters$Country, origin =
"iso2c", destination = "continent")
      submitters$submitterIdName <- paste(submitters$SubmitterID, submitters$Name, sep = " - ")

      ###################
      product <- r$dataBase$LastEcigProduct
      types <- r$dataBase$RefEcigProductType
      presentation <- r$dataBase$LastEcigPresentation
      sales <- r$dataBase$LastEcigSalesData

      # Define and fix colors for pie charts
      # From library pals : c(pals::stepped()[c(2, 6, 10, 14, 18, 22)], pals::tol()[c(6, 7)],
pals::watlington()[10])
      palette <- c("#B33E52", "#B3823E", "#78B33E", "#3E9FB3", "#653EB3", "#666666",
                   "#999933", "#DDCC77", "#9DAFFF")
      names(palette) <- unique(r$dataBase$RefEcigProductType$label)

      # paletteNico <- pals::brewer.paired(20)
      # names(paletteNico) <- rep(1:20)
      paletteNico <- pals::brewer.paired(21)
      names(paletteNico) <- rep(0:20)

      updateSelectizeInput(session = session, inputId = 'submitterId', choices =
submitters$submitterIdName, server = TRUE)

      observeEvent(input$submitterId,{

        if(!is.null(input$submitterId)){
          sub <- submitters[submitters$submitterIdName %in% input$submitterId, ]$SubmitterID

          output$boxSubmitters <- shinydashboard::renderInfoBox({
            valueBox(
              value = length(input$submitterId),
              subtitle = "Number of submitters",
              width = 2,
              color = "purple",
              icon = icon("user")
            )
          })

          output$boxProduct <- shinydashboard::renderInfoBox({
            valueBox(
              value = length(product[product$SubmitterID %in% sub, ]$ProductID),
              subtitle = "Number of products",
              width = 2,
              color = "blue",
              icon = icon("cube")
            )
          })
          output$boxPresentation <- shinydashboard::renderInfoBox({
            valueBox(
              value = length(presentation[presentation$SubmitterID %in% sub, ]$ProductID),
              subtitle = "Number of presentation",
              width = 2,
              color = "light-blue",
              icon = icon("cubes")
            )
          })

          ###### Product repartition :
          output$dwdpieChart <- downloadHandler(
            filename = function() {
              paste("ProductType", Sys.Date(), input$type, sep=".")
            },
            content = function(file) {
              if(input$type == "xlsx") {
                writexl::write_xlsx(product[product$SubmitterID %in% sub,], file)
              } else {
                write.csv(product[product$SubmitterID %in% sub,],
                          file= file,
                          row.names=F)
              }
            }
```

```
          )

          output$pieChart <- plotly::renderPlotly({
            tp <- as.data.frame(table(product[product$SubmitterID %in% sub,]$ProductType))
            colnames(tp) <- c("Type", "Quantity")
            tp$Type <- r$dataBase$RefEcigProductType[r$dataBase$RefEcigProductType$item %in%
tp$Type,]$label
            productType <- tp$Type
            qtt <- tp$Quantity
            fig <- plot_ly(tp, labels = ~Type, values = ~qtt, type = 'pie',
                           marker=list(colors = palette[tp$Type]))
            fig <- fig %>% layout(title = paste('Repartition of product type',
stringr::str_split(string = input$submitterId, pattern = "-")[[1]][2], sep = " - "),
                                  xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE),
                                  yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE))

          })
          ##########

          ###### Brand names :
          output$dwdBrandname <- downloadHandler(
            filename = function() {
              paste("BrandNames", Sys.Date(), input$typeBrandname, sep=".")
            },
            content = function(file) {
              if(input$typeBrandname == "xlsx") {
                writexl::write_xlsx(presentation[presentation$SubmitterID %in% sub, c("ProductID",
"LaunchDate", "BrandName","BrandSubtypeName")][input$brandname_rows_all,], file)
              } else {
                write.csv(presentation[presentation$SubmitterID %in% sub, c("ProductID",
"LaunchDate", "BrandName","BrandSubtypeName")][input$brandname_rows_all,],
                          file= file,
                          row.names=F)
              }
            }
          )

          output$brandname <- DT::renderDT({
            brandNameDf <- presentation[order(as.Date(presentation$LaunchDate, format =
"%Y%m/%d")),]
            DT::datatable(
              data = brandNameDf[brandNameDf$SubmitterID %in% sub, c("ProductID", "LaunchDate",
"BrandName","BrandSubtypeName")],
              rownames = FALSE,
              selection = "none",
              filter = 'top'
            )
          })
          ##########

          ###### nicotine :
          # typeNicotine
          output$dwdNicotine <- downloadHandler(
            filename = function() {
              paste("nicotineConcentration", Sys.Date(), input$typeNicotine, sep=".")
            },
            content = function(file) {

              filteredProduct <- subset(product, !is.na(NicotineConcentration))

              if(input$typeNicotine == "xlsx") {
                writexl::write_xlsx(filteredProduct[filteredProduct$SubmitterID %in% sub
                                                    & filteredProduct$NicotineConcentration <= 20, ], file)
              } else {
                write.csv(filteredProduct[filteredProduct$SubmitterID %in% sub
                                          & filteredProduct$NicotineConcentration <= 20, ],
                          file= file,
                          row.names=F)
              }
            }
          )

          output$nicotine <- plotly::renderPlotly({

            tp <- as.data.frame(table(round(product[product$SubmitterID %in% sub,
]$NicotineConcentration)))
```

```
            colnames(tp) <- c("Concentration", "Number of product")
            tp$Concentration <- as.numeric(as.character(tp$Concentration))

            tp <- tp[tp$Concentration <= 20,]
            tp$Concentration <- as.factor(tp$Concentration)

            concentration <- tp$Concentration
            qtt <- tp$`Number of product`

            print("concentration")
            print(table(concentration))

            tp %>%
              mutate(id = as.numeric(1)) %>%
              plot ly(x = ~id, y = ~qtt, color = ~concentration, colors =
paletteNico[levels(concentration)], type = 'bar', text = paste0("Concentration: ", concentration, "
mg/ml")) %>%
              layout(yaxis = list(title = 'qtt'), barmode = 'stack', xaxis = list(showticklabels =
FALSE))
          })
          ######

          ###### Sales :
          output$dwdSales <- downloadHandler(
            filename = function() {
              paste("SalesData", Sys.Date(), input$typeSales, sep=".")
            },
            content = function(file) {
              if(input$typeSales == "xlsx") {
                writexl::write_xlsx(sales[sales$SubmitterID %in% sub
                                          & sales$Year >= 2016
                                          & sales$Year <= as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]), ], file)
              } else {
                write.csv(sales[sales$SubmitterID %in% sub
                                & sales$Year >= 2016
                                & sales$Year <= as.integer(stringr::str_split(string = Sys.Date(),
pattern = "-")[[1]][1]), ],
                          file= file,
                          row.names=F)
              }
            }
          )

          output$salesData <- plotly::renderPlotly({

            if( all(sub %in% unique(sales$SubmitterID)) && length(sub) < 2){
              tmp <- sales[sales$SubmitterID %in% sub, c("Year", "SalesVolume")]

              # Filter and Clean Data
              tmp <- dplyr::filter(tmp, tmp$Year >= 2016)
              tmp <- dplyr::filter(tmp, tmp$Year <= as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]))

              if(!is.null(tmp)) {
                df <- stats::aggregate(.~Year, data = tmp, FUN = sum)
                fig <- plotly::plot_ly(df, x = ~Year, y = ~SalesVolume, name = 'SalesVolume', type =
'bar')
              } else {
                return(NULL)
              }

            } else {
              return(NULL)
            }

          })
        }

        # If no submitter selected :
################################################################################
        else {
          observeEvent(input$choice,{

            if(input$choice == "All"){

              output$regionChoice <- NULL
              output$countryChoice <- NULL
```

```
              updateSelectizeInput(session = session, inputId = 'submitterId', choices =
submitters$submitterIdName, server = TRUE)

              output$boxSubmitters <- shinydashboard::renderInfoBox({
                valueBox(
                  value = length(submitters$SubmitterID),
                  subtitle = "Number of submitters",
                  width = 2,
                  color = "purple",
                  icon = icon("user")
                )
              })

              output$boxProduct <- shinydashboard::renderInfoBox({
                valueBox(
                  value = length(product$ProductID),
                  subtitle = "Number of products",
                  width = 2,
                  color = "blue",
                  icon = icon("cube")
                )
              })

              output$boxPresentation <- shinydashboard::renderInfoBox({
                valueBox(
                  value = length(presentation$SubmitterID),
                  subtitle = "Number of presentations",
                  width = 2,
                  color = "light-blue",
                  icon = icon("cubes")
                )
              })


              ###### Product repartition :
              output$dwdpieChart <- downloadHandler(
                filename = function() {
                  paste("ProductType", Sys.Date(), input$type, sep=".")
                },
                content = function(file) {
                  if(input$type == "xlsx") {
                    writexl::write_xlsx(product, file)
                  } else {
                    write.csv(product,
                              file= file,
                              row.names=F)
                  }
                }
              )

              output$pieChart <- plotly::renderPlotly({
                tp <- as.data.frame(table(product$ProductType))
                colnames(tp) <- c("Type", "Quantity")
                tp$Type <- r$dataBase$RefEcigProductType[r$dataBase$RefEcigProductType$item %in%
tp$Type,]$label
                productType <- tp$Type
                qtt <- tp$Quantity

                fig <- plot_ly(tp, labels = ~Type, values = ~qtt, type = 'pie',
                               marker=list(colors = palette[tp$Type]))
                fig <- fig %>% layout(title = 'Global repartition of product type',
                                      xaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE),
                                      yaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE))
              })
              ##########

              ########## BrandNames :
              output$dwdBrandname <- downloadHandler(
                filename = function() {
                  paste("BrandNames", Sys.Date(), input$typeBrandname, sep=".")
                },
                content = function(file) {
                  if(input$typeBrandname == "xlsx") {
                    writexl::write_xlsx(presentation[ , c("ProductID", "LaunchDate",
"BrandName","BrandSubtypeName")][input$brandname_rows_all,], file)
                  } else {
```

```
                    write.csv(presentation[ , c("ProductID", "LaunchDate",
"BrandName","BrandSubtypeName")][input$brandname_rows_all,],
                              file= file,
                              row.names=F)
                  }
                }
              )

              output$brandname <- DT::renderDT({
                brandNameDf <- presentation[order(as.Date(presentation$LaunchDate, format =
"%Y%m/%d")),]
                DT::datatable(
                  brandNameDf[, c("ProductID", "LaunchDate","BrandName","BrandSubtypeName")],
                  rownames = FALSE,
                  selection = "none",
                  filter = 'top'
                )
              })
              #########


              ######### Nicotine :
              output$dwdNicotine <- downloadHandler(
                filename = function() {
                  paste("nicotineConcentration", Sys.Date(), input$typeNicotine, sep=".")
                },
                content = function(file) {

                  filteredProduct <- subset(product, !is.na(NicotineConcentration) &
NicotineConcentration <= 20)

                  if(input$typeNicotine == "xlsx") {
                    writexl::write_xlsx(filteredProduct, file)
                  } else {
                    write.csv(filteredProduct,
                              file= file,
                              row.names=F)

                  }
                }
              )

              output$nicotine <- plotly::renderPlotly({

                tp <- as.data.frame(table(round(product$NicotineConcentration)))
                colnames(tp) <- c("Concentration", "Number of product")
                tp$Concentration <- as.numeric(as.character(tp$Concentration))

                tp <- tp[tp$Concentration <= 20,]
                tp$Concentration <- as.factor(tp$Concentration)

                concentration <- tp$Concentration
                qtt <- tp$`Number of product`

                tp %>%
                  mutate(id = as.numeric(1)) %>%
                  plot_ly(x = ~id, y = ~qtt, color = ~concentration, colors =
paletteNico[levels(concentration)], type = 'bar', text = paste0("Concentration: ", concentration, "
mg/ml")) %>%
                  layout(yaxis = list(title = 'qtt'), barmode = 'stack', xaxis = list(showticklabels
= FALSE))
              })
              #########

              ######### Sales :
              output$dwdSales <- downloadHandler(
                filename = function() {
                  paste("SalesData", Sys.Date(), input$typeSales, sep=".")
                },
                content = function(file) {
                  if(input$typeSales == "xlsx") {
                    writexl::write_xlsx(sales[sales$Year >= 2016
                                              & sales$Year <= as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]), ], file)
                  } else {
                    write.csv(sales[sales$Year >= 2016
                                    & sales$Year <= as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]), ],
```

```
                              file= file,
                              row.names=F)

                    }
                  }
                )

                output$salesData <- plotly::renderPlotly({

                  # Filter and Clean Data
                  sales <- dplyr::filter(sales, sales$Year>=2016)
                  sales <- dplyr::filter(sales, sales$Year<=as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]))

                  sales <- sales[ ,c("Year", "SalesVolume")]
                  df <- stats::aggregate(.~Year, data = sales, FUN = sum)
                  fig <- plotly::plot_ly(df, x = ~Year, y = ~SalesVolume, name = 'SalesVolume', type =
'bar')
                })

              } else if(input$choice == "By Region"){
                output$countryChoice <- NULL
                output$regionChoice <- renderUI({
                  selectizeInput(inputId = ns("regionChoice"), label = "Select the region", choices =
unique(submitters$region), selected = "Europe", multiple = TRUE,
                                 options = list(
                                   placeholder = "Select a region",
                                   'plugins' = list('remove_button'),
                                   # 'create' = TRUE,
                                   'persist' = FALSE
                                 ))
                })

                ################################################################################

                observeEvent(input$regionChoice, {

                  sub_df <- submitters[submitters$region %in% input$regionChoice,]
                  updateSelectizeInput(session = session, inputId = 'submitterId', choices =
submitters[submitters$region %in% input$regionChoice,]$submitterIdName, server = TRUE)

                  output$boxSubmitters <- shinydashboard::renderInfoBox({
                    valueBox(
                      value = length(sub_df$SubmitterID),
                      subtitle = "Number of submitters",
                      width = 2,
                      color = "purple",
                      icon = icon("user")
                    )
                  })

                  output$boxProduct <- shinydashboard::renderInfoBox({
                    valueBox(
                      value = length(product[product$SubmitterID %in% sub_df$SubmitterID,]$ProductID),
                      subtitle = "Number of products",
                      width = 2,
                      color = "blue",
                      icon = icon("cube")
                    )
                  })

                  output$boxPresentation <- shinydashboard::renderInfoBox({
                    valueBox(
                      value = length(presentation[presentation$SubmitterID %in% sub_df$SubmitterID,
]$SubmitterID),
                      subtitle = "Number of presentations",
                      width = 2,
                      color = "light-blue",
                      icon = icon("cubes")
                    )
                  })

                  output$pieChart <- NULL

                  req(input$regionChoice)

                  ###### ZONE DE TEST
                  output$dwdpieChart <- downloadHandler(
```

```
                    filename = function() {
                      paste("ProductType", Sys.Date(), input$type, sep=".")
                    },
                    content = function(file) {
                      if(input$type == "xlsx") {
                        writexl::write_xlsx(product[product$SubmitterID %in% sub_df$SubmitterID,],
file)
                      } else {
                        write.csv(product[product$SubmitterID %in% sub_df$SubmitterID,],
                                  file= file,
                                  row.names=F)
                      }
                    }
                  )


                  output$pieChart <- plotly::renderPlotly({
                    tp <- as.data.frame(table(product[product$SubmitterID %in%
sub_df$SubmitterID,]$ProductType))
                    colnames(tp) <- c("Type", "Quantity")
                    tp$Type <- r$dataBase$RefEcigProductType[r$dataBase$RefEcigProductType$item %in%
tp$Type,]$label
                    productType <- tp$Type
                    qtt <- tp$Quantity
                    fig <- plot_ly(tp, labels = ~Type, values = ~qtt, type = 'pie',
                                   marker=list(colors = palette[tp$Type]))
                    fig <- fig %>% layout(title = paste('Repartition of product type',
input$regionChoice, sep = " - "),
                                          xaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE),
                                          yaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE))
                  })
                  ##########

                  ########## Brand Names :
                  output$dwdBrandname <- downloadHandler(
                    filename = function() {
                      paste("BrandNames", Sys.Date(), input$typeBrandname, sep=".")
                    },
                    content = function(file) {
                      if(input$typeBrandname == "xlsx") {
                        writexl::write_xlsx(presentation[presentation$SubmitterID %in%
sub_df$SubmitterID, c("ProductID", "LaunchDate",
"BrandName","BrandSubtypeName")][input$brandname_rows_all,], file)
                      } else {
                        write.csv(presentation[presentation$SubmitterID %in% sub_df$SubmitterID,
c("ProductID", "LaunchDate", "BrandName","BrandSubtypeName")][input$brandname_rows_all,],
                                  file= file,
                                  row.names=F)
                      }
                    }
                  )
                  output$brandname <- DT::renderDT({
                    brandNameDf <- presentation[order(as.Date(presentation$LaunchDate, format =
"%Y%m/%d")),]
                    DT::datatable(
                      brandNameDf[brandNameDf$SubmitterID %in% sub_df$SubmitterID, c("ProductID",
"LaunchDate","BrandName","BrandSubtypeName")],
                      rownames = FALSE,
                      selection = "none",
                      filter = 'top'
                    )
                  })
                  ##########

                  ########## Nicotine :
                  output$dwdNicotine <- downloadHandler(
                    filename = function() {
                      paste("nicotineConcentration", Sys.Date(), input$typeNicotine, sep=".")
                    },
                    content = function(file) {

                      filteredProduct <- subset(product, !is.na(NicotineConcentration))

                      if(input$typeNicotine == "xlsx") {
                        writexl::write_xlsx(filteredProduct[filteredProduct$SubmitterID %in%
sub_df$SubmitterID
```

```
                                                    & filteredProduct$NicotineConcentration <= 20, ],
file)
                    } else {
                      write.csv(filteredProduct[filteredProduct$SubmitterID %in% sub_df$SubmitterID
                                        & filteredProduct$NicotineConcentration <= 20, ],
                                file= file,
                                row.names=F)
                    }
                  }
                )

                output$nicotine <- plotly::renderPlotly({

                  tp <- as.data.frame(table(round(product[product$SubmitterID %in%
sub df$SubmitterID,]$NicotineConcentration)))
                  colnames(tp) <- c("Concentration", "Number of product")
                  tp$Concentration <- as.numeric(as.character(tp$Concentration))

                  tp <- tp[tp$Concentration <= 20,]
                  tp$Concentration <- as.factor(tp$Concentration)

                  concentration <- tp$Concentration
                  qtt <- tp$`Number of product`

                  tp %>%
                    mutate(id = as.numeric(1)) %>%
                    plot_ly(x = ~id, y = ~qtt, color = ~concentration, colors =
paletteNico[levels(concentration)], type = 'bar', text = paste0("Concentration: ", concentration, "
mg/ml")) %>%
                    layout(yaxis = list(title = 'qtt'), barmode = 'stack', xaxis =
list(showticklabels = FALSE))
                })

                ##########

                ########## Sales :
                output$dwdSales <- downloadHandler(
                  filename = function() {
                    paste("SalesData", Sys.Date(), input$typeSales, sep=".")
                  },
                  content = function(file) {
                    if(input$typeSales == "xlsx") {
                      writexl::write_xlsx(sales[sales$SubmitterID %in% sub_df$SubmitterID
                                                & sales$Year >= 2016
                                                & sales$Year <= as.integer(stringr::str_split(string
= Sys.Date(), pattern = "-")[[1]][1]), ], file)
                    } else {
                      write.csv(sales[sales$SubmitterID %in% sub_df$SubmitterID
                                      & sales$Year >= 2016
                                      & sales$Year <= as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]), ],
                                file= file,
                                row.names=F)
                    }
                  }
                )
                output$salesData <- plotly::renderPlotly({

                  sales <- sales[sales$SubmitterID %in% sub_df$SubmitterID, c("Year",
"SalesVolume")]
                  # Filter and Clean Data
                  sales <- dplyr::filter(sales, sales$Year>=2016)
                  sales <- dplyr::filter(sales, sales$Year<=as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]))

                  df <- stats::aggregate(.~Year, data = sales, FUN = sum)
                  fig <- plotly::plot_ly(df, x = ~Year, y = ~SalesVolume, name = 'SalesVolume', type
= 'bar')
                })

              }, ignoreNULL = FALSE)

            } else if(input$choice == "By Country"){

              output$regionChoice <- NULL
              output$countryChoice <- renderUI({

                # Get flags :
```

```
                img_urls <- paste0(
                  "https://cdn.rawgit.com/lipis/flag-icon-css/master/flags/4x3/",
                  tolower(unique(submitters$Country)), ".svg"
                )

                multiInput(
                  inputId = ns("pays"),
                  label = "Select the country : ",
                  choices = NULL,
                  selected = "France",
                  choiceNames = lapply(
                    seq_along(unique(submitters$countriesNames)),
                    function(i) {
                      tagList(
                        tags$img(src = img urls[i], width = 20, height = 15),
                        unique(submitters$countriesNames)[i]
                      )
                    }
                  ),
                  choiceValues = unique(submitters$countriesNames)
                )
              })


              observeEvent(input$pays, {

                sub_df <- submitters[submitters$countriesNames %in% input$pays,]

                updateSelectizeInput(session = session, inputId = 'submitterId', choices =
submitters[submitters$countriesNames %in% input$pays,]$submitterIdName, server = TRUE)

                output$boxSubmitters <- shinydashboard::renderInfoBox({
                  valueBox(
                    value = length(sub_df$SubmitterID),
                    subtitle = "Number of submitters",
                    width = 2,
                    color = "purple",
                    icon = icon("user")
                  )
                })

                output$boxProduct <- shinydashboard::renderInfoBox({
                  valueBox(
                    value = length(product[product$SubmitterID %in% sub_df$SubmitterID,]$ProductID),
                    subtitle = "Number of products",
                    width = 2,
                    color = "blue",
                    icon = icon("cube")
                  )
                })

                output$boxPresentation <- shinydashboard::renderInfoBox({
                  valueBox(
                    value = length(presentation[presentation$SubmitterID %in% sub_df$SubmitterID,
]$SubmitterID),
                    subtitle = "Number of presentations",
                    width = 2,
                    color = "light-blue",
                    icon = icon("cubes")
                  )
                })

                output$pieChart <- NULL
                req(input$pays)

                ###### ZONE DE TEST
                output$dwdpieChart <- downloadHandler(
                  filename = function() {
                    paste("ProductType", Sys.Date(), input$type, sep=".")
                  },
                  content = function(file) {
                    if(input$type == "xlsx") {
                      writexl::write_xlsx(product[product$SubmitterID %in% sub_df$SubmitterID,],
file)
                    } else {
                      write.csv(product[product$SubmitterID %in% sub_df$SubmitterID,],
                                file= file,
                                row.names=F)
```

```
                    }
                  }
                )

                output$pieChart <- plotly::renderPlotly({
                  tp <- as.data.frame(table(product[product$SubmitterID %in%
sub_df$SubmitterID,]$ProductType))
                  colnames(tp) <- c("Type", "Quantity")
                  tp$Type <- r$dataBase$RefEcigProductType[r$dataBase$RefEcigProductType$item %in%
tp$Type,]$label
                  productType <- tp$Type
                  qtt <- tp$Quantity
                  fig <- plot_ly(tp, labels = ~Type, values = ~qtt, type = 'pie',
                                 marker=list(colors = palette[tp$Type]))
                  fig <- fig %>% layout(title = paste('Repartition of product type', input$pays, sep
= " - "),
                                        xaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE),
                                        yaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE))
                })
                ##########

                ########## BrandName :
                output$dwdBrandname <- downloadHandler(
                  filename = function() {
                    paste("BrandNames", Sys.Date(), input$typeBrandname, sep=".")
                  },
                  content = function(file) {
                    if(input$typeBrandname == "xlsx") {
                      writexl::write_xlsx(presentation[presentation$SubmitterID %in%
sub_df$SubmitterID , c("ProductID", "LaunchDate",
"BrandName","BrandSubtypeName")][input$brandname_rows_all,], file)
                    } else {
                      write.csv(presentation[presentation$SubmitterID %in% sub_df$SubmitterID ,
c("ProductID", "LaunchDate", "BrandName","BrandSubtypeName")][input$brandname_rows_all,],
                                file= file,
                                row.names=F)
                    }
                  }
                )

                output$brandname <- DT::renderDT({
                  brandNameDf <- presentation[order(as.Date(presentation$LaunchDate, format =
"%Y%m/%d")),]
                  DT::datatable(
                    brandNameDf[brandNameDf$SubmitterID %in% sub_df$SubmitterID, c("ProductID",
"LaunchDate", "BrandName","BrandSubtypeName")],
                    rownames = FALSE,
                    selection = "none",
                    filter = 'top'
                  )
                })
                ##########

                ########## Nicotine :
                output$dwdNicotine <- downloadHandler(
                  filename = function() {
                    paste("nicotineConcentration", Sys.Date(), input$typeNicotine, sep=".")
                  },
                  content = function(file) {

                    filteredProduct <- subset(product, !is.na(NicotineConcentration))

                    if(input$typeNicotine == "xlsx") {
                      writexl::write_xlsx(filteredProduct[filteredProduct$SubmitterID %in%
sub_df$SubmitterID
                                                          & filteredProduct$NicotineConcentration <= 20, ],
file)
                    } else {
                      write.csv(filteredProduct[filteredProduct$SubmitterID %in% sub_df$SubmitterID
                                                & filteredProduct$NicotineConcentration <= 20, ],
                                file= file,
                                row.names=F)
                    }
                  }
                )
                output$nicotine <- plotly::renderPlotly({
```

```
                    tp <- as.data.frame(table(round(product[product$SubmitterID %in%
sub_df$SubmitterID,]$NicotineConcentration)))
                    colnames(tp) <- c("Concentration", "Number of product")
                    tp$Concentration <- as.numeric(as.character(tp$Concentration))

                    tp <- tp[tp$Concentration <= 20,]
                    tp$Concentration <- as.factor(tp$Concentration)

                    concentration <- tp$Concentration
                    qtt <- tp$`Number of product`

                    tp %>%
                      mutate(id = as.numeric(1)) %>%
                      plot_ly(x = ~id, y = ~qtt, color = ~concentration, colors =
paletteNico[levels(concentration)], type = 'bar', text = paste0("Concentration: ", concentration, "
mg/ml")) %>%
                      layout(yaxis = list(title = 'qtt'), barmode = 'stack', xaxis =
list(showticklabels = FALSE))
                  })
                  #############

                  ############# Sales :
                  output$dwdSales <- downloadHandler(
                    filename = function() {
                      paste("SalesData", Sys.Date(), input$typeSales, sep=".")
                    },
                    content = function(file) {
                      if(input$typeSales == "xlsx") {
                        writexl::write_xlsx(sales[sales$SubmitterID %in% sub_df$SubmitterID
                                                  & sales$Year >= 2016
                                                  & sales$Year <= as.integer(stringr::str_split(string
= Sys.Date(), pattern = "-")[[1]][1]), ], file)
                      } else {
                        write.csv(sales[sales$SubmitterID %in% sub_df$SubmitterID
                                        & sales$Year >= 2016
                                        & sales$Year <= as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]), ],
                                  file= file,
                                  row.names=F)
                      }
                    }
                  )

                  output$salesData <- plotly::renderPlotly({

                    sales <- sales[sales$SubmitterID %in% sub_df$SubmitterID, c("Year",
"SalesVolume")]
                    # Filter and Clean Data
                    sales <- dplyr::filter(sales, sales$Year>=2016)
                    sales <- dplyr::filter(sales, sales$Year<=as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]))

                    df <- stats::aggregate(.~Year, data = sales, FUN = sum)
                    fig <- plotly::plot_ly(df, x = ~Year, y = ~SalesVolume, name = 'SalesVolume', type
= 'bar')
                  })
                }, ignoreNULL = FALSE)
              } ## end last condition
            })# end observe event
          } # end else
        }, ignoreNULL = FALSE) # end observe envent submitter id

        t2 <- Sys.time()
        print("Temps sub : ")
        print(t2 - t1)

      }) #end observe
    }) # end module
} # end server

## To be copied in the UI
# mod_submitters_ui("submitters_1")

## To be copied in the server
# mod_submitters_server("submitters_1", r=r)
```

10. Source code: <TP> app_server.R

```
# app_server.R version 23-01-2024
#
# Copyright (c) Thomas El Khilali - euceg@anses.fr 2022-2023 (JATC2-WP5)
# Except specific files which bear a different mention, this programme is licensed under the EUPL-
1.2 or later
# You may obtain a copy of the license at https://joinup.ec.europa.eu/collection/eupl/eupl-text-
eupl-12
#
# This activity has received funding from the European Union's Health Program (2014-2020) under
grant agreement N°101035968 (JA-01-2020 - HP-JA-2020 - HP-JA-2020-2).
# The content of this document represents the views of the author only and is his/her sole
responsibility; it cannot be considered to reflect the views of the European Commission and/or the
European Health and Digital Executive
# Agency (HaDEA) or any other body of the European Union. The European Commission and the Agency do
not accept any responsibility for use that may be made of the information it contains.


# Increase maximum file size
options(shiny.maxRequestSize = 4000*1024^2)

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

  # when browser is closed, stop the terminal
  session$onSessionEnded(stopApp)

  # %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  # Data Importation
  # %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  mod_inputData_server(id = "inputData_1", r=r)


  # %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  # Submitters
  # %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  output$submittersMenu <- renderMenu({
    req(r$dataBase)
    menuItem(text = "Submitters", tabName = "submitters", icon = icon("user"))
  })

  output$presentationMenu <- renderMenu({
    req(r$dataBase)
    menuItem(text = "Presentations", tabName = "presentations", icon = icon("baby-formula", lib =
"glyphicon"))
  })

  output$ingredientsMenu <- renderMenu({
    req(r$dataBase)
    menuItem(text = "Ingredients", tabName = "ingredients", icon = icon("tint", lib = "glyphicon"))
  })

  mod_submitters_server(id = "submitters_1", r=r)
  mod_presentation_server(id = "presentation_1", r=r)
  mod_ingredients_server("ingredients_1", r=r)

}
```

11. Source code: <TP> app_ui.R

```
# app_ui.R version 23-01-2024
#
# Copyright (c) Thomas El Khilali - euceg@anses.fr 2022-2023 (JATC2-WP5)
# Except specific files which bear a different mention, this programme is licensed under the EUPL-
1.2 or later
# You may obtain a copy of the license at https://joinup.ec.europa.eu/collection/eupl/eupl-text-
eupl-12
#
# This activity has received funding from the European Union's Health Program (2014-2020) under
grant agreement N°101035968 (JA-01-2020 - HP-JA-2020 - HP-JA-2020-2).
# The content of this document represents the views of the author only and is his/her sole
responsibility; it cannot be considered to reflect the views of the European Commission and/or the
European Health and Digital Executive
# Agency (HaDEA) or any other body of the European Union. The European Commission and the Agency do
not accept any responsibility for use that may be made of the information it contains.


#' The application User-Interface
#'
#' @param request Internal parameter for `{shiny}`.
#'     DO NOT REMOVE.
#' @import shiny
#' @import shinythemes
#' @import shinydashboard
#'
#' @noRd
app_ui <- function(request) {

  tagList(
    # Leave this function for adding external resources
    golem_add_external_resources(),
    # Your application UI logic
    shinyjs::useShinyjs(),

    # To use icon from font awesome
    # tags$script(src = "https://kit.fontawesome.com/4afa144779.js"), #
https://fontawesome.com/v4/icons/

    # Main page :
    shinydashboard::dashboardPage(

      title = "EU-CEG Viewer",

      ## Header with logo and App title
      header = dashboardHeader(
        title = HTML("<div style = 'vertical-align:middle'>
                          <img src = 'www/jatc2.png' align = 'left' height = '100px'>
                          <h1>EU-CEG TP Viewer</h1>
                      </div>"
                     ), #<img src = 'img/anses.png' align = 'left' height = '100px'> to add for anses
logo
        titleWidth = "92%"
        # tags$li(class = "dropdown",
tags$a(href="https://github.com/ThomasElKhilali/dashboard_anses", icon("github"), "Repos"))
      ),

      # Sidebar with modules which appears once data is imported
      dashboardSidebar(
        sidebarMenu(
          id = "sidebar",
          menuItem(text = "Dataset", tabName = "data", icon = icon("database")),

          ## other menu wich appear when the database is loaded
          menuItemOutput("submittersMenu"),
          menuItemOutput("presentationMenu"),
          menuItemOutput("ingredientsMenu")
        )
      ),


      dashboardBody(

        # intern function for css header style
        headerCssStyle(),
```

```
        tabItems(
          # Onglet Data :
          mod_inputData_ui("inputData_1"),
          # Onglet Submission :
          mod_submitters_ui("submitters_1"),
          # Onglet presentation
          mod_presentation_ui("presentation_1"),
          # Onglet Ingredient
          mod_ingredients_ui("ingredients_1")

        )
      )
    )
  )
}

#' Add external Resources to the Application
#'
#' This function is internally used to add external
#' resources inside the Shiny application.
#'
#' @import shiny
#' @importFrom golem add_resource_path activate_js favicon bundle_resources
#' @noRd
golem_add_external_resources <- function() {

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

  tags$head(
    favicon(),
    bundle_resources(
      path = app_sys("app/www"),
      app_title = "eucegviewerTP"
    )

    # Add here other external resources
    # for example, you can add shinyalert::useShinyalert()
  )
}
```

12. Source code: <TP> fct_headerCssStyle.R

```
# fct_headerCssStyle.R version 23-01-2024
#
# Copyright (c) Thomas El Khilali - euceg@anses.fr 2022-2023 (JATC2-WP5)
# Except specific files which bear a different mention, this programme is licensed under the EUPL-1.2 or later
# You may obtain a copy of the license at https://joinup.ec.europa.eu/collection/eupl/eupl-text-eupl-12
#
# This activity has received funding from the European Union's Health Program (2014-2020) under grant agreement N°101035968 (JA-01-2020 - HP-JA-2020 - HP-JA-2020-2).
# The content of this document represents the views of the author only and is his/her sole responsibility; it cannot be considered to reflect the views of the European Commission and/or the European Health and Digital Executive
# Agency (HaDEA) or any other body of the European Union. The European Commission and the Agency do not accept any responsibility for use that may be made of the information it contains.

#' Header Css style
#' @noRd
headerCssStyle <- function(){

  tags$style(type="text/css", "
          /* https://jonkatz2.github.io/2018/06/22/Image-In-Shinydashboard-Header
          https://stackoverflow.com/questions/48978648/align-header-elements-in-shiny-dashboard */

          /*    Move everything below the header */
              .content-wrapper {
                  margin-top: 50px;
              }
              .content {
                  padding-top: 60px;
              }
          /*  Format the title/subtitle text */
              .title-box {
                  position: absolute;
                  text-align: center;
                  top: 50%;
                  left: 50%;
                  transform:translate(-50%, -50%);
              }
              @media (max-width: 590px) {
                  .title-box {
                      position: absolute;
                      text-align: center;
                      top: 10%;
                      left: 10%;
                      transform:translate(-5%, -5%);
                  }
              }

          /*    Make the image taller */
              .main-header .logo {
                  height: 125px;
              }
          /*    Override the default media-specific settings */
              @media (max-width: 5000px) {
                  .main-header {
                      padding: 0 0;
                      position: relative;
                  }
                  .main-header .logo,
                  .main-header .navbar {
                      width: 100%;
                      float: none;
                  }
                  .main-header .navbar {
                      margin: 0;
                  }
                  .main-header .navbar-custom-menu {
                      float: right;
                  }
              }
          /*    Move the sidebar down */
              .main-sidebar {
                  position: absolute;
```

```
                }
                .left-side, .main-sidebar {
                    padding-top: 175px;
                }"
    )
}
```

13. Source code: <TP> mod_ingredients.R

```
# mod_ingredients.R version 23-01-2024
#
# Copyright (c) Thomas El Khilali - euceg@anses.fr 2022-2023 (JATC2-WP5)
# Except specific files which bear a different mention, this programme is licensed under the EUPL-
1.2 or later
# You may obtain a copy of the license at https://joinup.ec.europa.eu/collection/eupl/eupl-text-
eupl-12
#
# This activity has received funding from the European Union's Health Program (2014-2020) under
grant agreement N°101035968 (JA-01-2020 - HP-JA-2020 - HP-JA-2020-2).
# The content of this document represents the views of the author only and is his/her sole
responsibility; it cannot be considered to reflect the views of the European Commission and/or the
European Health and Digital Executive
# Agency (HaDEA) or any other body of the European Union. The European Commission and the Agency do
not accept any responsibility for use that may be made of the information it contains.

#' ingredients UI Function
#'
#' @description A shiny Module.
#'
#' @param id,input,output,session Internal parameters for {shiny}.
#'
#' @noRd
#'
#' @importFrom shiny NS tagList
mod_ingredients_ui <- function(id){
  ns <- NS(id)
  tabItem(tabName = "ingredients",

          tabsetPanel(type = "tabs",
                      tabPanel("Substance stats",
                               fluidRow(
                                 box(title = "Ingredients selection", solidHeader = TRUE, status =
"primary", width = 12,
                                     selectizeInput(
                                       inputId = ns("ingredientsList"),
                                       label = "Ingredients :",
                                       multiple = FALSE,
                                       choices = NULL,
                                     )
                                 )
                               ),
                               fluidRow(

                                 shinydashboard::valueBoxOutput(ns("propProd"), width = 3),
                                 shinydashboard::valueBoxOutput(ns("propPres"), width = 3),
                                 ## Statut :
                                 box(title = "Informations", width = 6,
                                     solidHeader = TRUE, status = "primary", collapsible = TRUE,
                                     htmlOutput(ns("fonction")),
                                     htmlOutput(ns("toxicity")),
                                     textOutput(ns("onMarket"))
                                 ),

                               ),

                               fluidRow(
                                 box(title = "Product which contain the selected ingredient",
                                     solidHeader = TRUE, status = "primary", collapsible = TRUE,
                                     radioButtons(ns("typeproductOnMarket"), "Format type:",
                                                  choices = c(excel = "xlsx", csv = "csv"), inline =
TRUE),
                                     downloadButton(ns("dwdproductOnMarket"), "Download table",
                                                    style = "color: #fff; background-color: #27ae60;
border-color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                                     DTOutput(ns("productOnMarket"))
                                 ),
                                 box(title = "Summary of ingredient", collapsible = TRUE,
                                     solidHeader = TRUE, status = "primary",
                                     verbatimTextOutput(ns("summaryIngredients")),  ## Check values
??
                                     radioButtons(ns("typeboxplotIngredient"), "Format type:",
                                                  choices = c(excel = "xlsx", csv = "csv"), inline =
TRUE),
```

```
                                    downloadButton(ns("dwdboxplotIngredient"), "Download raw data",
                                                   style = "color: #fff; background-color: #27ae60;
border-color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                                    plotlyOutput(ns("boxplotIngredient"))
                                  )
                                )
                        ),
                        tabPanel("Generic stats",
                                fluidRow(
                                  box(title = "Ingredients selection", solidHeader = TRUE, status =
"primary", width = 12,
                                      selectizeInput(
                                        inputId = ns("parentIngredientList"),
                                        label = "Ingredients :",
                                        multiple = FALSE,
                                        choices = NULL,
                                      )
                                  )
                                ),
                                fluidRow(

                                  shinydashboard::valueBoxOutput(ns("ParentpropProd"), width = 3),
                                  shinydashboard::valueBoxOutput(ns("ParentpropPres"), width = 3),
                                  ## Statut :
                                  box(title = "Informations", width = 6,
                                      solidHeader = TRUE, status = "primary", collapsible = TRUE,
                                      htmlOutput(ns("Parentfonction")),
                                      htmlOutput(ns("Parenttoxicity")),
                                      textOutput(ns("ParentonMarket"))
                                  ),

                                ),

                                fluidRow(
                                  box(title = "Product which contain the selected ingredient",
                                      solidHeader = TRUE, status = "primary", collapsible = TRUE,
                                      radioButtons(ns("typeParentproductOnMarket"), "Format type:",
                                                   choices = c(excel = "xlsx", csv = "csv"), inline =
TRUE),
                                      downloadButton(ns("dwdParentproductOnMarket"), "Download
table",
                                                     style = "color: #fff; background-color: #27ae60;
border-color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                                      DTOutput(ns("ParentproductOnMarket"))
                                  ),
                                  box(title = "Summary of ingredient", collapsible = TRUE,
                                      solidHeader = TRUE, status = "primary",
                                      verbatimTextOutput(ns("ParentsummaryIngredients")),  ## Check
values ??
                                      radioButtons(ns("typeParentboxplotIngredient"), "Format type:",
                                                   choices = c(excel = "xlsx", csv = "csv"), inline =
TRUE),
                                      downloadButton(ns("dwdParentboxplotIngredient"), "Download raw
data",
                                                     style = "color: #fff; background-color: #27ae60;
border-color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                                      plotlyOutput(ns("ParentboxplotIngredient"))
                                  )
                                )
                        )
                ),
  )
}

#' ingredients Server Functions
#'
#' @importFrom stringi stri_detect_fixed
#' @importFrom stats ave
#' @importFrom utils write.csv
#' @importFrom data.table setDT
#'
#' @noRd
mod_ingredients_server <- function(id, r){
  moduleServer( id, function(input, output, session){

    ns <- session$ns

    observe({
```

```
      req(r$dataBase)

      t1 <- Sys.time()

      ## get data :
      sub <- r$dataBase$LastTobaccoSubmission
      ingredients <- r$dataBase$LastTobaccoIngredientOther
      recip <- r$dataBase$LastTobaccoRecipeSubst
      map <- r$dataBase$MapIngSubst
      mapFunct <- r$dataBase$MapTobaccoIngOtherCountFunction
      product <- r$dataBase$LastTobaccoProduct
      presentation <- r$dataBase$LastTobaccoPresentation
      submitters <- r$dataBase$LastTobaccoSubmitterDetails
      RefIngredientFunction <- r$dataBase$RefIngredientFunction
      RefToxicityStatus <- r$dataBase$RefToxicityStatus

      # Format recip :
      # library(data.table)
      data.table::setDT(recip)
      data.table::setDT(mapFunct)
      recip <- merge(recip, mapFunct, by = c("SubmissionID", "IngredientCount")) # add ingredient
function
      recip <-  merge(recip, ingredients[, c("SubmissionID", "IngredientCount", "ToxicityStatus")],
by = c("SubmissionID", "IngredientCount")) # add ingredient tox status

      recip[is.na((recip$GenericSubstName)),]$SubstID <- "IN_PROGRESS"
      recip[is.na((recip$GenericSubstName)),]$SubstCAS <- "IN_PROGRESS"
      recip[is.na((recip$GenericSubstName)),]$SubstName <- "IN_PROGRESS"
      recip[is.na((recip$GenericSubstName)),]$GenericSubstName <- "IN_PROGRESS"
      recip$RecipeQuantity[recip$RecipeQuantity < 0] <- 0

      # Dataframe to get the list of ingredients
      df <- as.data.frame(table(recip$SubstCAS))
      colnames(df) <- c("CASNumber", "Frequency")
      df <- df %>% arrange(desc(df$Frequency))
      df$name <- recip[match(x = df$CASNumber, table = recip$SubstCAS ),]$SubstName
      df$ParentSubstName <- recip[match(x = df$CASNumber, table = recip$SubstCAS
),]$GenericSubstName
      ####

      ##############################################################################################

      # verifier si acceleré
      updateSelectizeInput(session = session, inputId = "ingredientsList", choices =
unique(paste(df$CASNumber, df$name, sep = " ")), server = TRUE)
      updateSelectizeInput(session = session, inputId = "parentIngredientList", choices =
unique(recip$GenericSubstName), server = TRUE)

      ## New to check
      observeEvent(input$ingredientsList, {
        print(input$ingredientsList)
        # Save cas number of the ingredient
        cas <- strsplit(x = input$ingredientsList, split = " ")[[1]][1] # get the cas of the
ingredient
        # print(cas)

        # Use of CAS to compute the number of product who has the ingredient :
        proportionProduct <- (length(unique(recip[recip$SubstCAS == cas, ]$ProductID)) /
                                length(unique(recip$ProductID))) * 100


        proportionPresentation <- (dim(presentation[presentation$ProductID %in%
unique(recip[recip$SubstCAS == cas, ]$ProductID), ])[[1]] /
                                     dim(presentation)[[1]]) * 100
        # General stat :
        output$propProd <- shinydashboard::renderInfoBox({
          valueBox(
            value = paste0(round(x = proportionProduct, digits = 3), " %"),
            subtitle = "products",
            width = 2,
            color = "blue",
            icon = icon("cube")
          )
        })
        output$propPres <- shinydashboard::renderInfoBox({
          valueBox(
            value = paste0(round(x = proportionPresentation, digits = 3), " %"),
```

```
            subtitle = "presentation",
            width = 2,
            color = "light-blue",
            icon = icon("cubes")
          )
        })

        # Get the function of the substance :
        funct <- names(which.max(table(recip[recip$SubstCAS == cas, ]$Function)))
        output$fonction <- renderText({
          paste0("The function of the selected product is mostly classified as <b>",
                 RefIngredientFunction[RefIngredientFunction$item == funct, ]$label_en, "</b>")
        })

        # Get the toxicity status of the substance :
        tox <- names(which.max(table(recip[recip$SubstCAS == cas, ]$ToxicityStatus)))
        output$toxicity <- renderText({
          paste0("Most declared toxicity : <b>",
                 RefToxicityStatus[RefToxicityStatus$item == tox, ]$label_en, "</b>")
        })

        # Check if the product is currently on the market
        onMarket <- NULL
        if("active" %in% sub[unique(recip[recip$SubstCAS == cas, ]$ProductID) %in%
sub$ProductID,]$List){
          output$onMarket <- renderText({
            paste0("Ingredient present in products on the market")
          })

          ##### Product repartition :
          output$dwdproductOnMarket <- downloadHandler(
            filename = function() {
              paste("ProductOnMarket", Sys.Date(), input$typeproductOnMarket, sep=".")
            },
            content = function(file) {
              if(input$typeproductOnMarket == "xlsx") {
                writexl::write_xlsx(sub[sub$ProductID %in% unique(recip[recip$SubstCAS == cas,
]$ProductID),
                                        c("SubmitterID", "ProductID", "LastUpdate",
"List")][input$productOnMarket_rows_all,], file)
              } else {
                write.csv(sub[sub$ProductID %in% unique(recip[recip$SubstCAS == cas, ]$ProductID),
                              c("SubmitterID", "ProductID", "LastUpdate",
"List")][input$productOnMarket_rows_all,],
                          file= file,
                          row.names=F)
              }
            }
          )

          output$productOnMarket <- renderDT({
            datatable(
              data = sub[sub$ProductID %in% unique(recip[recip$SubstCAS == cas, ]$ProductID),
                         c("SubmitterID", "ProductID", "LastUpdate", "List")],
              rownames = FALSE,
              filter = 'top',
              selection = "none",
              options = list(
                dom = 'Bfrtip',
                deferRender = TRUE,
                scrollY = 400,
                scroller = TRUE
              )
            )

          })


        } else {
          output$onMarket <- renderText({
            paste0("Ingredient not in products on the market")
          })
        }

        ## Box plot for summary of the selected ingredients
        output$summaryIngredients <- renderPrint({
          summary(recip[recip$SubstCAS == cas, ]$RecipeConcPPM)
        })
```

```
        ###### Ingredient data dwd:
        output$dwdboxplotIngredient <- downloadHandler(
          filename = function() {
            paste("ingredientSummary", Sys.Date(), input$typeboxplotIngredient, sep=".")
          },
          content = function(file) {
            if(input$typeboxplotIngredient == "xlsx") {
              writexl::write_xlsx(recip[recip$SubstCAS == cas, ], file)
            } else {
              write.csv(recip[recip$SubstCAS == cas, ],
                        file= file,
                        row.names=F)
            }
          }
        )
        ## Graphical representation of the summary information
        output$boxplotIngredient <- renderPlotly({
          plot_ly(data = recip[recip$SubstCAS == cas, ], y = ~RecipeConcPPM, type = "box", name =
"", hoverinfo = "y") %>%
            layout(title = paste0("Boxplot of ", unique(recip[recip$SubstCAS == cas, ]$SubstCAS)),
                   yaxis = list(type = "log"), xaxis = list(visible = FALSE))
        })
      })


      ### For the second onglet parents
      observeEvent(input$parentIngredientList, {

        req(input$parentIngredientList)

        cas <- strsplit(x = input$parentIngredientList, split = " ")[[1]][1] # get the cas of the
ingredient

        # Use of CAS to compute the number of product who has the ingredient
        proportionProduct <- (length(unique(recip[recip$GenericSubstName ==
input$parentIngredientList, ]$ProductID)) /
                                length(unique(ingredients$ProductID))) * 100

        proportionPresentation <- (dim(presentation[presentation$ProductID %in%
unique(recip[recip$GenericSubstName == input$parentIngredientList, ]$ProductID), ])[[1]] /
                                     dim(presentation)[[1]]) * 100

        # Get the function of the substance :
        funct <- names(which.max(table(recip[recip$GenericSubstName == input$parentIngredientList,
]$Function)))

        output$Parentfonction <- renderText({
          paste0("The function of the selected product is mostly classified as <b>",
                 RefIngredientFunction[RefIngredientFunction$item == funct, ]$label_en, "</b>")
        })

        # Get the toxicity status of the substance :
        tox <- names(which.max(table(recip[recip$GenericSubstName == input$parentIngredientList,
]$ToxicityStatus)))
        output$Parenttoxicity <- renderText({
          paste0("Most declared toxicity : <b>",
                 RefToxicityStatus[RefToxicityStatus$item == tox, ]$label_en, "</b>")
        })

        output$ParentpropProd <- shinydashboard::renderInfoBox({
          valueBox(
            value = paste0(round(x = proportionProduct, digits = 3), " %"),
            subtitle = "products",
            width = 2,
            color = "blue",
            icon = icon("cube")
          )
        })
        output$ParentpropPres <- shinydashboard::renderInfoBox({
          valueBox(
            value = paste0(round(x = proportionPresentation, digits = 3), " %"),
            subtitle = "presentation",
            width = 2,
            color = "light-blue",
            icon = icon("cubes")
          )
        })
```

```
        # Check if the ingredient is in a product currently on the market.
        onMarket <- NULL
        if("active" %in% sub[unique(recip[recip$GenericSubstName == input$parentIngredientList,
]$ProductID) %in% sub$ProductID,]$List){
          output$ParentonMarket <- renderText({
            paste0("Ingredient present in products on the market")
          })

          ###### Ingredient data dwd:
          output$dwdParentproductOnMarket <- downloadHandler(
            filename = function() {
              paste("ingredientSummary", Sys.Date(), input$typeParentproductOnMarket, sep=".")
            },
            content = function(file) {
              if(input$typeParentproductOnMarket == "xlsx") {
                writexl::write_xlsx(sub[sub$ProductID %in% unique(recip[recip$GenericSubstName ==
input$parentIngredientList, ]$ProductID),
                                        c("SubmitterID", "ProductID", "LastUpdate",
"List")][input$ParentproductOnMarket_rows_all,], file)

              } else {
                write.csv(sub[sub$ProductID %in% unique(recip[recip$SubstCAS == cas, ]$ProductID),
                              c("SubmitterID", "ProductID", "LastUpdate",
"List")][input$productOnMarket_rows_all,],
                          file= file,
                          row.names=F)
              }
            }
          )

          output$ParentproductOnMarket <- renderDT({
            datatable(
              data = sub[sub$ProductID %in% unique(recip[recip$GenericSubstName ==
input$parentIngredientList, ]$ProductID),
                         c("SubmitterID", "ProductID", "LastUpdate", "List")],
              rownames = FALSE,
              filter = 'top',selection = "none",
              options = list(
                dom = 'Bfrtip',
                deferRender = TRUE,
                scrollY = 400,
                scroller = TRUE
              )
            )

          })
        } else {
          output$ParentonMarket <- renderText({
            paste0("Ingredient not in products on the market")
          })
        }

        ## Box plot for summary of the selected ingredients
        output$ParentsummaryIngredients <- renderPrint({
          summary(recip[recip$GenericSubstName == input$parentIngredientList, ]$RecipeConcPPM)
        })

#########################################################################################################
############

        # Download the data of the boxplot
        output$dwdParentboxplotIngredient <- downloadHandler(
          filename = function() {
            paste("ProductType", Sys.Date(), input$typeParentboxplotIngredient, sep=".")
          },
          content = function(file) {
            if(input$typeParentboxplotIngredient == "xlsx") {
              writexl::write_xlsx(recip[recip$GenericSubstName == input$parentIngredientList, ],
file)
            } else {
              write.csv(recip[recip$GenericSubstName == input$parentIngredientList, ],
                        file= file,
                        row.names=F)
            }
          }
        )
```

```
        ## Graphical representation of the summary information
        output$ParentboxplotIngredient <- renderPlotly({
          plot_ly(data = recip[recip$GenericSubstName == input$parentIngredientList, ], y=
~RecipeConcPPM, type = "box", name = "", hoverinfo = "y") %>%
            layout(title = paste0("Boxplot of ", unique(recip[recip$GenericSubstName ==
input$parentIngredientList, ]$SubstCAS)),
                   yaxis = list(type = "log"), xaxis = list(visible = FALSE))
        })

      })

      t2 <- Sys.time()
      print("Temps ing : ")
      print(t2 - t1)

    })
  })
}

## To be copied in the UI
# mod_ingredients_ui("ingredients_1")

## To be copied in the server
# mod_ingredients_server("ingredients_1")
```

14. Source code: <TP> mod_inputData.R

```
# mod_inputData.R version 23-01-2024
#
# Copyright (c) Thomas El Khilali - euceg@anses.fr 2022-2023 (JATC2-WP5)
# Except specific files which bear a different mention, this programme is licensed under the EUPL-
1.2 or later
# You may obtain a copy of the license at https://joinup.ec.europa.eu/collection/eupl/eupl-text-
eupl-12
#
# This activity has received funding from the European Union's Health Program (2014-2020) under
grant agreement N°101035968 (JA-01-2020 - HP-JA-2020 - HP-JA-2020-2).
# The content of this document represents the views of the author only and is his/her sole
responsibility; it cannot be considered to reflect the views of the European Commission and/or the
European Health and Digital Executive
# Agency (HaDEA) or any other body of the European Union. The European Commission and the Agency do
not accept any responsibility for use that may be made of the information it contains.


#' inputData UI Function
#'
#' @description Module to import the database into the app
#'
#' @param id,input,output,session Internal parameters for {shiny}.
#'
#' @importFrom shinyFiles shinyFilesButton getVolumes shinyFileChoose
#' @importFrom DT DTOutput
#' @import shinydashboard
#' @importFrom utils write.csv
#' @importFrom shinyBS bsTooltip
#'
#' @noRd
#'
#' @importFrom shiny NS tagList
mod_inputData_ui <- function(id){
  ns <- NS(id)

  tabItem(tabName = "data",
          tags$head(tags$style(".shiny-notification {position: fixed; top: 60% ;left: 50%")),
          tabBox(id = "t1", width = 12,
                 # tabPanel(title = "About", icon=icon("address-card"),
                 #          h2("About this tool"),
                 #          tags$p("This app aims to provide a tool to analyse and describe
                 #                 data from producer declaration available on EU-CEG.",br(),
                 #                 "Sed mauris nisi, sollicitudin nec rutrum a, vestibulum vitae
orci.
                 #                 Mauris at suscipit tellus. Pellentesque habitant morbi tristique
senectus
                 #                 et netus et malesuada fames ac turpis egestas. In vehicula est
                 #                 lectus, eget tincidunt purus sodales non. Ut ultricies, velit
                 #                 sollicitudin volutpat porttitor, ligula neque posuere nibh,
                 #                 ornare ullamcorper urna eros quis erat. Ut hendrerit mauris nec
                 #                 elementum congue. Praesent in ex nisi."
                 #          ),
                 #          h2("Contact us"),
                 #          tags$p("If you have troubles using the app or if you have any
                 #                  questions or feedback relating to the data or the tool, then
                 #                 please contact us at xxxxxxx@anses.fr and we will be happy to
help.")
                 # ),
                 tabPanel(title = "Data",icon = icon("table"),
                          shinyFiles::shinyFilesButton(ns("Btn_GetFile"), "Choose a file" ,
                                                       title = "Please select a file:", multiple =
FALSE,
                                                       buttonType = "default", class = NULL),
                          # actionButton(ns("get"), "Choose the database"),

                          tags$i(
                             class = "glyphicon glyphicon-info-sign",
                             style = "color:#0072B2;",
                             title = "Please, choose the database file format .db"
                          ),


                          shinyBS::bsTooltip(id = ns("Btn_GetFile"), title = "Please, choose the
database file format .db",
```

```
                                placement = "left", trigger = "hover"),
                        tags$div(uiOutput(ns("formatDownload")),
                                 style= "display:inline-block; margin-right:2.5%; margin-
left:2.5%"),
                        tags$div(uiOutput(ns("buttonDownload")),
                                 style ="display:inline-block; margin-right:2.5%; margin-
left:2.5%"),
                        uiOutput(ns("listTables")),
                        DT::DTOutput(ns("dataFrame"))

                )
        )
  )
}

#' inputData Server Functions
#'
#' @import DBI
#' @import RSQLite
#' @importFrom DT renderDT datatable
#' @importFrom writexl write_xlsx
#' @importFrom shinyBS addTooltip
#' @importFrom shinyFiles getVolumes shinyFileChoose parseFilePaths
#'
#' @noRd
mod_inputData_server <- function(id, r){
  moduleServer(id, function(input, output, session){
    ns <- session$ns

    volumes = shinyFiles::getVolumes()

    dataBase <- list()

    # val = reactiveVal()

    # observeEvent(input$get, {
    #   val(file.choose())
    # })

      observe({

        # req(val())

        # shinyjs::toggle(id = "type", anim = FALSE, animType = "slide", time = 0.5, selector =
NULL, condition = r$dataBase)
        shinyFiles::shinyFileChoose(input = input,id = "Btn_GetFile",
                                    roots = volumes, session = session, filetypes= "db")

        file_selected <- shinyFiles::parseFilePaths(roots = volumes,
                                                    selection =  req(input$Btn_GetFile))

        # output$txt_file <- renderText(as.character(req(file_selected$datapath)))

        # Open the link to the SQL data base
        db <- DBI::dbConnect(RSQLite::SQLite(), as.character(req(file_selected$datapath)))

        # db <- DBI::dbConnect(RSQLite::SQLite(), as.character(val()))

        # tmp filter to speed up tests : less table = less loading time
        filter <- c("LastSubmitterDetails","LastTobaccoAttachment", "LastTobaccoEmission",
                    "LastTobaccoEntityDetails", "LastTobaccoSubmitterHierarchy",
"RefSubmissionType",
                    "RefTobaccoOtherIngredientCategory","RefTobaccoPackageType",
                    "RefToxicologicalDataAvailable","SubmitterDetails","TobaccoAttachment",
                    "TobaccoEmission", "TobaccoEntityDetails","TobaccoIngNotMap",
"TobaccoIngredient",
                    "TobaccoIngredientOther", "TobaccoPresentation",
"TobaccoProduct","TobaccoRecipe" ,
                    "TobaccoRecipeStats","TobaccoRecipeSubst","TobaccoSubmission",
"TobaccoSubmitterHierarchy", "IngList")
        # keep : "LastTobaccoSalesData","LastTobaccoProduct", "LastTobaccoSubmitterDetails",
        # "RefTobaccoProductType", "MapIngSubst","TobaccoSalesData", "LastTobaccoSubmission",
        # "MapEcigIngCountFunction" , "LastTobaccoSubmission", "LastTobaccoPresentation",
"LastTobaccoIngredient",
        # "RefTobaccoLeafType", "RefTobaccoPartType","RefToxicityStatus","RefTobaccoLeafCureMethod",
        # "LastTobaccoIngredientOther",
"LastTobaccoRecipeSubst","MapIngSubst","MapTobaccoIngOtherCountFunction",
        # "RefIngredientFunction","DbInfo",
```

```
        # adapte names for the structur depending on the number of loaded tables
        names <- DBI::dbListTables(db)[!DBI::dbListTables(db) %in% filter]

        # Progress bar
        withProgress(message = 'Loading...', value = 0, {

          n <- length(DBI::dbListTables(db)) - length(filter) + 1 # number of time we go through the
loop

          # Go through tables :
          for(tbl in DBI::dbListTables(db)){
            # ... exept the one chosen in the filter
            if(! tbl %in% filter){
              incProgress(1/n,
                          detail = paste("Importing ", tbl)) # progress bar
              print(tbl) # dbg
              # Read the table
              df <- DBI::dbReadTable(conn = db, name = tbl)
              # Store it into the database structure
              dataBase[[length(dataBase)+1]]<- df
            } # end if
          } # end for

          names(dataBase) <- names # give names of each list component
        }) # end progress Bar

        DBI::dbDisconnect(conn = db) # disconnect the link to the database

        ### Visualisation :
        # Select the table to display
        output$listTables <- renderUI({
          selectInput(inputId = ns("tables"), label = "Select tables", choices =
names(dataBase[!startsWith(names(dataBase), "Ref")]))
        })


        ## To download data :
        output$dwd <- downloadHandler(
          filename = function() {
            paste(input$tables, Sys.Date(), input$type, sep=".")
          },
          content = function(file) {
            if(input$type == "xlsx") {
              writexl::write_xlsx(dataBase[[input$tables]][input$dataFrame_rows_all,], file)
            } else {
              write.csv(dataBase[[input$tables]][input$dataFrame_rows_all,],
                        file= file,
                        row.names=F)
            }
          }
        )

        # Display raw tables once they are loaded
        output$dataFrame <- DT::renderDT({

          req(input$tables)

          DT::datatable( ## to do : continue to add buttons to ease the use of dt (download
button...)
            data = as.data.frame(dataBase[input$tables]),
            filter = 'top',
            rownames = FALSE,
            fillContainer = TRUE,
            class = "compact hover stripe",
            selection = "none",
            options = list(pageLength = 10, autoWidth = TRUE, scrollX=TRUE, scrollY = "400px",
scrollCollapse=TRUE)
            )

        })

        r$dataBase <- dataBase # store database for other modules (global scope throughr$dataBase)

        ## test success message after importation
        if(!is.null(dataBase)){ ## r$dataBase
          output$formatDownload <- renderUI(
            radioButtons(ns("type"), "Format type:",
```

```
                        choices = c(excel = "xlsx", csv = "csv"), inline = TRUE)
          )
          output$buttonDownload <- renderUI(
            downloadButton(ns("dwd"), "Download table",
                           style = "color: #fff; background-color: #27ae60; border-color:
#fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;")
          )

          ## Possible to delay ??
          sendSweetAlert(
            session = session,
            title = "Data imported",
            text = "You may have to wait a minute before going further",
            type = "success"
          )
        }
    }) # end observe
  }) # end module
} # end server

## To be copied in the UI
# mod_inputData_ui("inputData_1")

## To be copied in the server
# mod_inputData_server("inputData_1")
```

15. Source code: <TP> mod_presentation.R

```
# mod_presentation.R version 23-01-2024
#
# Copyright (c) Thomas El Khilali - euceg@anses.fr 2022-2023 (JATC2-WP5)
# Except specific files which bear a different mention, this programme is licensed under the EUPL-
1.2 or later
# You may obtain a copy of the license at https://joinup.ec.europa.eu/collection/eupl/eupl-text-
eupl-12
#
# This activity has received funding from the European Union's Health Program (2014-2020) under
grant agreement N°101035968 (JA-01-2020 - HP-JA-2020 - HP-JA-2020-2).
# The content of this document represents the views of the author only and is his/her sole
responsibility; it cannot be considered to reflect the views of the European Commission and/or the
European Health and Digital Executive
# Agency (HaDEA) or any other body of the European Union. The European Commission and the Agency do
not accept any responsibility for use that may be made of the information it contains.


#' presentation UI Function
#'
#' @description A shiny Module.
#'
#' @param id,input,output,session Internal parameters for {shiny}.
#'
#' @noRd
#'
#' @importFrom shiny NS tagList
mod_presentation_ui <- function(id){
  ns <- NS(id)
  tabItem(tabName = "presentations",
          fluidPage(
            # Icon colors :
            tags$style(".okicon {color:#008000}"),
            tags$style(".removeicon {color:#FF0000}"),
            tags$style(".hourglassicon {color:#FFA500}"),

            fluidRow(
              box(title = "Filters", solidHeader = TRUE, width = 8, status = "primary",
                  collapsible = TRUE,

                  column(width = 4,
                         selectizeInput(
                           inputId = ns("submitterId"),
                           label = "Submitter :",
                           multiple = FALSE,
                           choices = NULL
                         )
                  ),

                  column(width = 4,
                         selectizeInput(
                           inputId = ns("brandName"),
                           label = "Brand Name :",
                           multiple = FALSE,
                           # choices = paste(presentation$BrandName, presentation$BrandSubtypeName,
sep = " "),
                           choices = NULL
                         )
                  )
              ),

              box(title = "Informations", solidHeader = TRUE, width = 4, collapsible = TRUE, status
= "primary",
                  htmlOutput(ns("nbProducts")),
                  htmlOutput(ns("productType")),
                  htmlOutput(ns("nicotineConcentration"))
              )


            ),
            #UI FIGURES
            fluidRow(
              column(width = 2,
                     selectizeInput(
                       inputId = ns("relatedProducts"),
```

```
                        label = "Related products :",
                        multiple = FALSE,
                        choices = NULL
                      ),

              ),
              column(width = 10,
                     shinydashboard::valueBoxOutput(ns("firstSubmissionDateBox"), width = 3),
                     shinydashboard::valueBoxOutput(ns("lastestSubmissionDateBox"), width = 3)
              ),
            ),
            fluidRow(
              box(title = "Presentation related to the selected product", solidHeader = TRUE,
collapsible = TRUE, status = "primary",
                  width = 8,
                  tags$div(radioButtons(ns("typeRelatedProductsDf"), "Format type:",
                                        choices = c(excel = "xlsx", csv = "csv"), inline = TRUE),
                           style ="display:inline-block; margin-right:2.5%; margin-left:2.5%"),
                  tags$div(downloadButton(ns("dwdRelatedProductsDf"), "Download table",
                                          style = "color: #fff; background-color: #27ae60; border-
color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                           style ="display:inline-block; margin-right:2.5%; margin-left:2.5%"),
                  DTOutput(ns("relatedProductsDf"))
              ),
              box(title = "Icons", solidHeader = TRUE, collapsible = TRUE, status = "primary",
                  width = 4,
                  icon("ok", lib = "glyphicon", class = "okicon"),"The presentation is currently on
the market",br(),
                  icon("stop", lib = "glyphicon"),"The presentation was on the market but not
anymore",br(),
                  icon("remove", lib = "glyphicon", class = "removeicon", verify_fa = FALSE), "The
presentation never has been on the market",br(),
                  icon("hourglass", lib = "glyphicon", class = "hourglassicon"), "The presentation
is planned to be release on the market"
              )
            ),
            fluidRow(
              div(style = "background-color: #f2f2f2; height: 100%;",
                  div(style = "display: flex; align-items: center; justify-content: center; height:
100%;",
                      div(style = "background-color: #c4d4e8; border: 1px solid #ccc; padding: 20px;
max-width: 500px; text-align: center; border-radius: 10px; position: relative;",
                          p("To display sales data information and the list of ingredients, please
select a row in the dataframe above. It will select a presentation and show its informations"),
                          div(style = "font-size: 32px; color: black; background-color: #f2f2f2;
width: 50px; height: 50px; border-radius: 50%; display: flex; align-items: center; justify-content:
center; position: absolute; bottom: -25px; left: 50%; transform: translateX(-50%); border: 2px solid
black;",
                              HTML("&#8595;")
                          )
                      )
                  )
              ),
              tags$br(),
              br(),
              br()
            ),
            fluidRow(
              shinydashboard::valueBoxOutput(ns("launchDateBox"), width = 3),
              shinydashboard::valueBoxOutput(ns("withdrawalDate"), width = 3)
              # shinydashboard::valueBoxOutput(ns("nicotineConcentration"), width = 3)
            ),

            fluidRow(
              box(title = "Presentation Sales Data", solidHeader = TRUE, #width = 10,
                  collapsible = TRUE, status = "primary",
                  textOutput(ns("noSalesData")),
                  plotly::plotlyOutput(ns("prensetationSalesDataDF"))
              ),
              box(title = "Presentation Ingredients", solidHeader = TRUE,  #width = 10,
                  collapsible = TRUE, status = "primary",
                  tags$div(radioButtons(ns("typeIngredientsDf"), "Format type:",
                                        choices = c(excel = "xlsx", csv = "csv"), inline = TRUE),
                           style ="display:inline-block; margin-right:2.5%; margin-left:2.5%"),
                  tags$div(downloadButton(ns("dwdIngredientsDf"), "Download table",
                                          style = "color: #fff; background-color: #27ae60; border-
color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                           style ="display:inline-block; margin-right:2.5%; margin-left:2.5%"),
```

```
                DTOutput(ns("ingredientsDT"))
              )
            )
          )
  )
}

#' presentation Server Functions
#'
#' @noRd
mod_presentation_server <- function(id, r){
  moduleServer( id, function(input, output, session){

    ns <- session$ns

    observe({

      req(r$dataBase)

      t1 <- Sys.time()

      # Format submitters table :
      submitters <- r$dataBase$LastTobaccoSubmitterDetails
      # Cleaning
      submitters$Country <- stringr::str_replace_all(string = submitters$Country, pattern=" ",
repl="")
      # Mapping countries names with full name and region
      submitters$countriesNames <- countrycode::countrycode(sourcevar = submitters$Country, origin =
"iso2c", destination = "country.name")
      submitters$region <- countrycode::countrycode(sourcevar = submitters$Country, origin =
"iso2c", destination = "continent")
      submitters$submitterIdName <- paste(submitters$SubmitterID, submitters$Name, sep = " - ")

      # Get Submission
      submission <- r$dataBase$LastTobaccoSubmission
      RefTobaccoProductType <- r$dataBase$RefTobaccoProductType

      # load submitter list from the server side (big list)
      updateSelectizeInput(session = session, inputId = 'submitterId', choices = c("None",
submitters$submitterIdName), selected = "None", server = TRUE)
      # updateSelectizeInput(session = session, inputId = 'productType', choices = c("All",
unique(RefTobaccoProductType$label)), selected = "All", server = TRUE)

      # Get sales :
      sales <- r$dataBase$LastTobaccoSalesData

      # Get product
      product <- r$dataBase$LastTobaccoProduct

      # Get presentation
      presentation <- r$dataBase$LastTobaccoPresentation
      # Format brand Names to improve browse :
      presentation$combinedBrandNames <- paste(presentation$BrandName,
presentation$BrandSubtypeName, sep = " ")
      presentation$combinedBrandNames <- gsub(pattern = " NA", replacement = "", x =
presentation$combinedBrandNames) # remove NA to clean names
      presentation$ProductType <- product$ProductType[match(presentation$ProductID,
product$ProductID)]
      presentation$ProductTypeName <- RefTobaccoProductType$label_en[match(presentation$ProductType,
RefTobaccoProductType$item)]
      presentation$status <- submission$List[match(presentation$ProductID, submission$ProductID)]

      # Get ingredients and refs
      ingredients <- r$dataBase$LastTobaccoRecipeSubst
      # RefTobaccoPartType <- r$dataBase$RefTobaccoPartType
      # RefTobaccoLeafType <- r$dataBase$RefTobaccoLeafType
      # RefTobaccoLeafCureMethod <- r$dataBase$RefTobaccoLeafCureMethod

      ## Mapping :
      # ingredients$PartType <- RefTobaccoPartType$label_en[match(ingredients$PartType,
RefTobaccoPartType$item)]
      # ingredients$LeafType <- RefTobaccoLeafType$label_en[match(ingredients$LeafType,
RefTobaccoLeafType$item)]
      # ingredients$LeafCureMethod <-
RefTobaccoLeafCureMethod$label_en[match(ingredients$LeafCureMethod, RefTobaccoLeafCureMethod$item)]
      ###

      # Drop List of brand names
```

```
      updateSelectizeInput(session = session, inputId = 'brandName', choices = c("None",
presentation$combinedBrandNames), selected = "None", server = TRUE)

      df <- data.frame()

      observeEvent(input$submitterId,{
        req(input$submitterId)
        # If a submitter is selected : change the type and brand name
        if(input$submitterId == "None") {
          updateSelectizeInput(session = session, inputId = 'productType', choices = c("All",
unique(RefTobaccoProductType$label)), selected = "All", server = TRUE)
          updateSelectizeInput(session = session, inputId = 'brandName', choices = c("None",
presentation$combinedBrandNames), selected = "None", server = TRUE)
        } else {
          id <- submitters[submitters$submitterIdName == input$submitterId,]$SubmitterID

          tp <- as.data.frame(table(product[product$SubmitterID %in% id,]$ProductType))
          colnames(tp) <- c("Type", "Quantity")
          tp$Type <- RefTobaccoProductType[RefTobaccoProductType$item %in% tp$Type,]$label

          updateSelectizeInput(session = session, inputId = 'productType', choices = c("All",
unique(tp$Type)), selected = "All", server = TRUE)

          # Update brandnames :
          brandNms <- presentation[presentation$SubmitterID == id, ]$combinedBrandNames
          updateSelectizeInput(session = session, inputId = 'brandName', choices = c("None",
unique(brandNms)), selected = "None", server = TRUE)

        }

      })

      ## Pour la suite once it is filtered :
      observeEvent(input$brandName, {
        req(input$brandName)

        ## Reset ingredients DT when brandName change.

        if(input$brandName != "None") {

          # Update related product depending of the selected brandname
          updateSelectizeInput(session = session, inputId = "relatedProducts", choices =
unique(presentation[presentation$combinedBrandNames == input$brandName, ]$ProductID), server = TRUE)

          output$nbProducts <- renderText({

            paste("This brandname presentation was found in <b>",
                  length(unique(presentation[presentation$combinedBrandNames == input$brandName,
]$ProductID)),
                  " product(s) </b> <I>(check the droplist Related Product below)</I>")
          })

          output$productType <- renderText({
            paste("The selected presentation is declared as <b>",
                  unique(presentation[presentation$combinedBrandNames == input$brandName
                                      & presentation$ProductID == input$relatedProducts,
]$ProductTypeName,"</b>")
            )
          })

          output$lastestSubmissionDateBox <- shinydashboard::renderInfoBox({
            # presentation <- presentation[presentation$ProductID == input$relatedProducts, ]
            # strsplit(x = , split = " ")[[1]][1]
            req(input$relatedProducts)
            valueBox(
              value = strsplit(x = submission[submission$ProductID == input$relatedProducts,
]$LastUpdate,
                               split = " ")[[1]][1], # Get only the date wich is the first element
because the string is too long
              subtitle = "Latest Submission",
              width = 2,
              color = "teal",
              icon = icon("arrows-rotate")
            )
          })

          output$firstSubmissionDateBox <- shinydashboard::renderInfoBox({
            req(input$relatedProducts)
```

```
            valueBox(
              value = strsplit(x = submission[submission$ProductID == input$relatedProducts,
]$FirstSubmission,
                               split = " ")[[1]][1], # Get only the date wich is the first element
because the string is too long
              subtitle = "First Submission",
              width = 2,
              color = "teal",
              icon = icon("export", lib = "glyphicon")
            )
          })



          # output$nicotineConcentration <- shinydashboard::renderInfoBox({
          #   # presentation <- presentation[presentation$ProductID == input$relatedProducts, ]
          #   valueBox(
          #     value = paste(unique(product[product$ProductID == input$relatedProducts,
]$NicotineConcentration), " mg/mL"),
          #     subtitle = "Nicotine concentration",
          #     width = 2,
          #     color = "maroon",
          #     # icon = icon("remove", verify_fa = FALSE)
          #   )
          # })

          output$nicotineConcentration <- renderText({
            paste("Nicotine concentration : <b>",
                  unique(product[product$ProductID == input$relatedProducts, ]$Nicotine),
                  " mg/mL</b>")
          })

          output$relatedProductsDf <- renderDT({
            req(input$relatedProducts)
            df <<- presentation[presentation$ProductID == input$relatedProducts, ]

            df <<- dplyr::select(df, c("PresentationCount", "combinedBrandNames", "ProductTypeName",
"LaunchDate", "WithdrawalDate", "status", "ProductID"))
            df$LaunchDate <<- as.Date(df$LaunchDate)
            df$WithdrawalDate <<- as.Date(df$WithdrawalDate)

            ## Check all dates to see if the presentation is available or not
            df$state <<- apply(X = df, MARGIN = 1, function(x) {
              # For each line of the DT, store values
              launch <- x[4] # get launchdate in df
              withdraw <- x[5] # get withdrawal date in df
              status <- x[6] # get status date in df

              if(!is.na(withdraw)){
                if((withdraw < launch) || (launch == withdraw && withdraw < as.Date(Sys.time()))) {
                  ico <- as.character(icon("remove", lib = "glyphicon", class = "removeicon",
verify_fa = FALSE)) #"Presentation removed from the market"
                } else if ((withdraw > launch && withdraw < as.Date(Sys.time()))) {
                  ico <- as.character(icon("stop", lib = "glyphicon")) #"Presentation removed from
the market"
                }
                else if (withdraw > launch && withdraw > as.Date(Sys.time()) && launch <
as.Date(Sys.time())) {
                  ico <- as.character(icon("ok", lib = "glyphicon", class = "okicon"))#"On market"
                } else if((launch == withdraw && launch > as.Date(Sys.time())) ||
                          (withdraw > launch && launch > as.Date(Sys.time()))) {
                  ico <- as.character(icon("hourglass", lib = "glyphicon", class =
"hourglassicon"))#"On market"
                }
              } else { # OK
                if(launch < as.Date(Sys.time())){
                  # ico <- as.character(icon("ok", lib = "glyphicon", class = "okicon"))#"On market"
                  if(!status == "inactive"){
                    ico <- as.character(icon("ok", lib = "glyphicon", class = "okicon"))#"On market"
                  } else {
                    ico <- as.character(icon("stop", lib = "glyphicon"))
                  }
                } else {
                  ico <- as.character(icon("hourglass", lib = "glyphicon", class =
"hourglassicon"))#"On market"
                }
              }
              # #hourglass for sablier (waiting)
```

```
            return(ico)
          })

          DT::datatable(
            data = dplyr::select(df, c("PresentationCount", "combinedBrandNames", "LaunchDate",
"WithdrawalDate", "state")),#df, "ProductTypeName",
            rownames = FALSE,
            escape = FALSE,
            selection = list(mode = 'single')
          )
        })

        output$dwdRelatedProductsDf <- downloadHandler(
          filename = function() {
            paste("otherPresentation",Sys.Date(), input$typeRelatedProductsDf, sep=".")
          },
          content = function(file) {
            if(input$typeRelatedProductsDf == "xlsx") {
              writexl::write_xlsx(dplyr::select(df, c("PresentationCount", "combinedBrandNames",
"LaunchDate", "WithdrawalDate", "state")), file)
            } else {
              write.csv(dplyr::select(df, c("PresentationCount", "combinedBrandNames",
"LaunchDate", "WithdrawalDate", "state")),
                        file= file,
                        row.names=F)
            }
          }
        )

        output$salesDataProduct <- plotly::renderPlotly({

          req(input$relatedProducts)

          # Filter the dates
          sales <- dplyr::filter(sales, sales$Year>=2016)
          sales <- dplyr::filter(sales, sales$Year<=as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]))

          # Select the product
          sales <- sales[sales$ProductID == input$relatedProducts, c("Year", "SalesVolume")]

          if(nrow(sales) > 0){
            dfSales <- stats::aggregate(.~Year, data = sales, FUN = sum)
            fig <- plotly::plot_ly(dfSales, x = ~Year, y = ~SalesVolume, name = 'SalesVolume',
type = 'bar')
            }

        })

        observeEvent(input$relatedProductsDf_rows_selected,{
          req(input$relatedProductsDf_rows_selected)

          # vector with values of the selected line
          row <- df[input$relatedProductsDf_rows_selected, ]

          output$launchDateBox <- shinydashboard::renderInfoBox({
            valueBox(
              value = row$LaunchDate,
              subtitle = "Launch date",
              width = 2,
              color = "green",
              icon = icon("calendar")
            )
          })

          output$withdrawalDate <- shinydashboard::renderInfoBox({
            req(input$relatedProducts)
            valueBox(
              value = row$WithdrawalDate,
              subtitle = "Withdrawal date",
              width = 2,
              color = "red",
              icon = icon("remove", verify_fa = FALSE)
            )
          })

          selectedPresentationSales <- sales[which( sales$ProductID == row$ProductID &
```

```
                                                    sales$PresentationCount ==
row$PresentationCount), c("Year", "SalesVolume")]

            if(dim(selectedPresentationSales)[1] == 0){
              output$prensetationSalesDataDF <- NULL
              output$noSalesData <- renderText({
                print("No presentation sales data")
              })
            } else {
              if(sum(selectedPresentationSales$SalesVolume) == 0){
                output$prensetationSalesDataDF <- NULL
                output$noSalesData <- renderText({
                  print("Declared sales data is null")
                })
              } else { # if sales data is available
                output$noSalesData <- NULL
                output$prensetationSalesDataDF <- renderPlotly({
                  fig <- plotly::plot_ly(selectedPresentationSales, x = ~Year, y = ~SalesVolume,
name = 'SalesVolume', type = 'bar')
                  fig %>% layout(xaxis = list(range = c(2015, as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]))))
                })

              }

            }

            output$ingredientsDT <- renderDT({
              ingTable <- ingredients[ingredients$ProductID == row$ProductID, ] # c("CasClean",
"UpName", "Quantity")

              datatable(
                # ingTable[order(ingTable$Quantity,decreasing=TRUE), c("TobaccoIngredientCount",
"PartType",
                #                                                     "LeafType", "LeafCureMethod",
"Quantity")],
                ingTable[, c("Name", "RecipeQuantity")],
                rownames = FALSE,
                selection = "none",
                filter = 'top'
              )

            })
            ## To download data :
            output$dwdIngredientsDf <- downloadHandler(
              filename = function() {
                paste("ingredientsList",Sys.Date(), input$typeIngredientsDf, sep=".")
              },
              content = function(file) {
                if(input$typeIngredientsDf == "xlsx") {
                  writexl::write_xlsx(ingredients[ingredients$ProductID == row$ProductID,
c("TobaccoIngredientCount", "PartType",

"LeafType", "LeafCureMethod", "Quantity")][input$ingredientsDT_rows_all,], file)
                } else {
                  write.csv(ingredients[ingredients$ProductID == row$ProductID,
c("TobaccoIngredientCount", "PartType",
                                                                                    "LeafType",
"LeafCureMethod", "Quantity")][input$ingredientsDT_rows_all,],
                            file= file,
                            row.names=F)
                }
              }
            )

          })

        } else {
          output$lastestSubmissionDateBox <- shinydashboard::renderInfoBox({
            valueBox(
              value = "None",
              subtitle = "Latest Submission",
              width = 2,
              color = "teal",
              icon = icon("arrows-rotate")
            )
          })
          output$firstSubmissionDateBox <- shinydashboard::renderInfoBox({
```

```
            valueBox(
              value = "None",
              subtitle = "First Submission",
              width = 2,
              color = "teal",
              icon = icon("export", lib = "glyphicon")
            )
          })

          output$nbProducts <- renderText({
            paste("This brandname presentation was found in ", 0,  " product(s)")

          })
          output$productType <- renderText({
            paste("The selected presentation is ...")
          })

          output$launchDateBox <- shinydashboard::renderInfoBox({
            valueBox(
              value = "None",
              subtitle = "Launch date",
              width = 2,
              color = "green",
              icon = icon("calendar")
            )
          })
          output$withdrawalDate <- shinydashboard::renderInfoBox({
            valueBox(
              value = "None",
              subtitle = "Withdrawal date",
              width = 2,
              color = "red",
              icon = icon("remove", verify_fa = FALSE)
            )
          })

          output$nicotineConcentration <- renderText({
            paste("Nicotine concentration : <b>",
                  0,
                  " mg/mL</b>")
          })
          output$ingredientsDT <- NULL
        }

      })

      # output$productDF <- renderDT({
      #   datatable(
      #     dfBrowser,
      #     plugins = "ellipsis",
      #     filter = 'top',
      #     selection = list(mode = 'single'),
      #     rownames = FALSE,
      #     options = list(
      #       columnDefs = list(list(
      #         targets = c(1, 2, 3, 4, 5, 6, 7, 8),
      #         render = DT::JS("$.fn.dataTable.render.ellipsis( 50, false )")
      #       )))
      #   )
      # })

      observeEvent(input$productDF_rows_selected,{
        req(input$productDF_rows_selected)

        # row <- dfBrowser[input$productDF_rows_selected, ] # vector with values of the selected
line
        # Contain :
        # "SubmissionID"       "ProductID"          "Name"               "BrandName"
"BrandSubtypeName"
        # "combinedBrandNames" "NationalComment"    "Description"        "GeneralComment"

        ######################################### Change names :
        # Update related products based on
        output$relatedProductsBrowse <- renderUI({
          selectizeInput(inputId = ns("relatedProductsBrowse"), label = "Related products :",
multiple = FALSE, choices = unique(presentation[presentation$combinedBrandNames %in% row[6],
]$ProductID))
        })
```

```
        ########################################
        output$lastestSubmissionDateBoxBrowse <- shinydashboard::renderInfoBox({
          presentation <- presentation[presentation$ProductID %in% row[2], ]
          valueBox(
            value = min(unique(presentation[presentation$combinedBrandNames %in% row[6],
]$LaunchDate)),
            subtitle = "Latest Submission",
            width = 2,
            color = "teal",
            icon = icon("calendar")
          )
        })

        output$firstSubmissionBoxBrowse <- NULL

        output$launchDateBoxBrowse <- shinydashboard::renderInfoBox({
          presentation <- presentation[presentation$ProductID %in% input$relatedProductsBrowse, ]
          valueBox(
            value = min(unique(presentation[presentation$combinedBrandNames %in% row[6],
]$LaunchDate)),
            subtitle = "Launch date",
            width = 2,
            color = "green",
            icon = icon("calendar")
          )
        })

        output$withdrawalDateBrowse <- shinydashboard::renderInfoBox({
          presentation <- presentation[presentation$ProductID %in% input$relatedProductsBrowse, ]
          valueBox(
            value = unique(presentation[presentation$combinedBrandNames %in% row[6],
]$WithdrawalDate),
            subtitle = "Withdrawal date",
            width = 2,
            color = "red",
            icon = icon("remove", verify_fa = FALSE)
          )
        })

        output$nicotineConcentration <- renderText({
          paste("Nicotine concentration : <b>",
                unique(product[product$ProductID %in% row[2], ]$Nicotine),
                " mg/mL</b>")
        })


      })

      t2 <- Sys.time()
      print("Temps pres : ")
      print(t2 - t1)

    })

  })
}

## To be copied in the UI
# mod_presentation_ui("presentation_1")

## To be copied in the server
# mod_presentation_server("presentation_1")
```

16. Source code: <TP> mod_submitters.R

```
# mod_submitters.R version 23-01-2024
#
# Copyright (c) Thomas El Khilali - euceg@anses.fr 2022-2023 (JATC2-WP5)
# Except specific files which bear a different mention, this programme is licensed under the EUPL-
1.2 or later
# You may obtain a copy of the license at https://joinup.ec.europa.eu/collection/eupl/eupl-text-
eupl-12
#
# This activity has received funding from the European Union's Health Program (2014-2020) under
grant agreement N°101035968 (JA-01-2020 - HP-JA-2020 - HP-JA-2020-2).
# The content of this document represents the views of the author only and is his/her sole
responsibility; it cannot be considered to reflect the views of the European Commission and/or the
European Health and Digital Executive
# Agency (HaDEA) or any other body of the European Union. The European Commission and the Agency do
not accept any responsibility for use that may be made of the information it contains.


#' submitters UI Function
#'
#' @description A shiny Module.
#'
#' @param id,input,output,session Internal parameters for {shiny}.
#'
#' @import shinyWidgets
#' @import plotly
#' @import shinycssloaders
#' @importFrom shiny NS tagList
#'
#' @noRd
#'
mod_submitters_ui <- function(id){
  ns <- NS(id)
  tabItem(tabName = "submitters",
          fluidRow(
            box(title = "Filters", solidHeader = TRUE, status = "primary", width = 10,
                column(width = 3,
                       awesomeRadio(
                         inputId = ns("choice"),
                         label = "Submitter origin",
                         choices = c("All", "By Region","By Country"),
                         selected = "All",
                         checkbox = FALSE,
                         status = "primary"
                       ),
                ),
                column(width = 3,
                       uiOutput(ns("regionChoice")),
                       uiOutput(ns("countryChoice")),
                ),
                column(width = 4,
                       selectizeInput(
                         inputId = ns("submitterId"),
                         label = "Submitter :",
                         multiple = TRUE,
                         choices = NULL,
                         options = list(
                           placeholder = "Select a submitter by ID or name",
                           'plugins' = list('remove_button'),
                           'persist' = FALSE
                         )
                       )
                )
            )
          ),
          fluidRow(
            shinydashboard::valueBoxOutput(ns("boxSubmitters"), width = 4),
            shinydashboard::valueBoxOutput(ns("boxProduct"), width = 4),
            shinydashboard::valueBoxOutput(ns("boxPresentation"), width = 4)
          ),
          fluidRow(
            box(collapsible = TRUE, solidHeader = TRUE, title = "Product repartition", status =
"primary",
                tags$div(radioButtons(ns("type"), "Format type:",
                                      choices = c(excel = "xlsx", csv = "csv"), inline = TRUE),
```

```
                        style= "display:inline-block; margin-right:2.5%; margin-left:2.5%"
                ),
                tags$div(downloadButton(ns("dwdpieChart"), "Download raw data",
                                        style = "color: #fff; background-color: #27ae60; border-
color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                        style= "display:inline-block; margin-right:2.5%; margin-left:2.5%"
                ),
                shinycssloaders::withSpinner(plotly::plotlyOutput(outputId = ns("pieChart")))
            ),
            box(collapsible = TRUE, solidHeader = TRUE,title = "Brand Names", status = "primary",
                tags$div(radioButtons(ns("typeBrandname"), "Format type:",
                                      choices = c(excel = "xlsx", csv = "csv"), inline = TRUE),
                        style= "display:inline-block; margin-right:2.5%; margin-left:2.5%"
                ),
                tags$div(downloadButton(ns("dwdBrandname"), "Download table",
                                        style = "color: #fff; background-color: #27ae60; border-
color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                        style= "display:inline-block; margin-right:2.5%; margin-left:2.5%"
                ),
                DT::DTOutput(ns("brandname"))
            )
          ),
          fluidRow(
            box(collapsible = TRUE, solidHeader = TRUE, title = "Sales Data", status = "primary",
                tags$div(radioButtons(ns("typeSales"), "Format type:",
                                      choices = c(excel = "xlsx", csv = "csv"), inline = TRUE),
                        style= "display:inline-block; margin-right:2.5%; margin-left:2.5%"
                ),
                tags$div(downloadButton(ns("dwdSales"), "Download raw data",
                                        style = "color: #fff; background-color: #27ae60; border-
color: #fff;padding: 5px 14px 5px 14px;margin: 5px 5px 5px 5px;"),
                        style= "display:inline-block; margin-right:2.5%; margin-left:2.5%"
                ),
                shinycssloaders::withSpinner(plotly::plotlyOutput(outputId = ns("salesData")))
            )
          )
  )
}

#' submitters Server Functions
#'
#' @importFrom countrycode countrycode
#' @importFrom stringr str_replace_all str_split
#' @import dplyr
#' @importFrom grDevices rainbow
#' @importFrom stats aggregate
#' @importFrom utils write.csv
#' @import pals
#'
#' @noRd
mod_submitters_server <- function(id, r){
  moduleServer( id, function(input, output, session){
    ns <- session$ns

    # Need observe to access r$dataBase
    observe({

      req(r$dataBase) # doesn't run if no data imported

      t1 <- Sys.time() # to get an idea of time spent

      #### Data preparation

      # Format submitters table :
      submitters <- r$dataBase$LastTobaccoSubmitterDetails
      # Cleaning
      submitters$Country <- stringr::str_replace_all(string = submitters$Country, pattern = " ",
repl = "")
      # Mapping countries names with full name and region
      submitters$countriesNames <- countrycode::countrycode(sourcevar = submitters$Country, origin =
"iso2c", destination = "country.name")
      submitters$region <- countrycode::countrycode(sourcevar = submitters$Country, origin =
"iso2c", destination = "continent")
      submitters$submitterIdName <- paste(submitters$SubmitterID, submitters$Name, sep = " - ")

      ###################
      product <- r$dataBase$LastTobaccoProduct
      RefTobaccoProductType <- r$dataBase$RefTobaccoProductType
```

```
      presentation <- r$dataBase$LastTobaccoPresentation
      sales <- r$dataBase$LastTobaccoSalesData

      # Define and fix colors for pie charts - product repartition and nicotine concentration
      # From library pals : c(pals::stepped()[c(2, 6, 10, 14, 18, 22)], pals::tol()[c(6, 7)],
pals::watlington()[10])
      palette <- c("#B33E52", "#B3823E", "#78B33E", "#3E9FB3", "#653EB3", "#666666",
                   "#999933", "#DDCC77", "#9DAFFF", "#0040FF", "#A901DB", "#2EFE9A")
      names(palette) <- unique(RefTobaccoProductType$label)

      paletteNico <- pals::brewer.paired(20)
      names(paletteNico) <- rep(1:20)

      updateSelectizeInput(session = session, inputId = 'submitterId', choices =
submitters$submitterIdName, server = TRUE)

      observeEvent(input$submitterId,{

        if(!is.null(input$submitterId)){
          sub <- submitters[submitters$submitterIdName %in% input$submitterId, ]$SubmitterID

          output$boxSubmitters <- shinydashboard::renderInfoBox({
            valueBox(
              value = length(input$submitterId),
              subtitle = "Number of submitters",
              width = 2,
              color = "purple",
              icon = icon("user")
            )
          })

          output$boxProduct <- shinydashboard::renderInfoBox({
            valueBox(
              value = length(product[product$SubmitterID %in% sub, ]$ProductID),
              subtitle = "Number of products",
              width = 2,
              color = "blue",
              icon = icon("cube")
            )
          })
          output$boxPresentation <- shinydashboard::renderInfoBox({
            valueBox(
              value = length(presentation[presentation$SubmitterID %in% sub, ]$ProductID),
              subtitle = "Number of presentation",
              width = 2,
              color = "light-blue",
              icon = icon("cubes")
            )
          })

          ###### Product repartition :
          output$dwdpieChart <- downloadHandler(
            filename = function() {
              paste("ProductType", Sys.Date(), input$type, sep=".")
            },
            content = function(file) {
              if(input$type == "xlsx") {
                writexl::write_xlsx(product[product$SubmitterID %in% sub,], file)
              } else {
                write.csv(product[product$SubmitterID %in% sub,],
                          file= file,
                          row.names=F)
              }
            }
          )

          output$pieChart <- plotly::renderPlotly({
            tp <- as.data.frame(table(product[product$SubmitterID %in% sub,]$ProductType))
            colnames(tp) <- c("Type", "Quantity")
            tp$Type <- RefTobaccoProductType[RefTobaccoProductType$item %in% tp$Type,]$label
            productType <- tp$Type
            qtt <- tp$Quantity
            fig <- plot_ly(tp, labels = ~Type, values = ~qtt, type = 'pie',
                           marker=list(colors = palette[tp$Type]))
            fig <- fig %>% layout(title = paste('Repartition of product type',
stringr::str_split(string = input$submitterId, pattern = "-")[[1]][2], sep = " - "),
                                  xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE),
```

```
                                yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE))

          })
          ##########

          ###### Brand names :
          output$dwdBrandname <- downloadHandler(
            filename = function() {
              paste("BrandNames", Sys.Date(), input$typeBrandname, sep=".")
            },
            content = function(file) {
              if(input$typeBrandname == "xlsx") {
                writexl::write_xlsx(presentation[presentation$SubmitterID %in% sub, c("ProductID",
"LaunchDate", "BrandName","BrandSubtypeName")][input$brandname_rows_all,], file)
              } else {
                write.csv(presentation[presentation$SubmitterID %in% sub, c("ProductID",
"LaunchDate", "BrandName","BrandSubtypeName")][input$brandname_rows_all,],
                          file= file,
                          row.names=F)
              }
            }
          )

          output$brandname <- DT::renderDT({
            brandNameDf <- presentation[order(as.Date(presentation$LaunchDate, format =
"%Y%m/%d")),]
            DT::datatable(
              data = brandNameDf[brandNameDf$SubmitterID %in% sub, c("ProductID", "LaunchDate",
"BrandName","BrandSubtypeName")],
              rownames = FALSE,
              selection = "none",
              filter = 'top'
            )
          })
          ##########

          ###### nicotine :
          # typeNicotine
          # output$dwdNicotine <- downloadHandler(
          #   filename = function() {
          #     paste("nicotineConcentration", Sys.Date(), input$typeNicotine, sep=".")
          #   },
          #   content = function(file) {
          #     if(input$typeNicotine == "xlsx") {
          #       writexl::write_xlsx(product[product$SubmitterID %in% sub
          #                                  & product$Nicotine <= 20, ], file)
          #     } else {
          #       write.csv(product[product$SubmitterID %in% sub
          #                         & product$Nicotine <= 20, ],
          #                 file= file,
          #                 row.names=F)
          #     }
          #   }
          # )
          #
          # output$nicotine <- plotly::renderPlotly({
          #
          #   tp <- as.data.frame(table(round(product[product$SubmitterID %in% sub, ]$Nicotine)))
          #   colnames(tp) <- c("Concentration", "Number of product")
          #   tp$Concentration <- as.numeric(tp$Concentration)
          #
          #   tp <- tp[tp$Concentration <= 20,]
          #   tp$Concentration <- as.factor(tp$Concentration)
          #
          #   concentration <- tp$Concentration
          #   qtt <- tp$`Number of product`
          #
          #   tp %>%
          #     mutate(id = as.numeric(1)) %>%
          #     plot_ly(x = ~id, y = ~qtt, color = ~concentration, colors =
paletteNico[concentration], type = 'bar') %>%
          #     layout(yaxis = list(title = 'qtt'), barmode = 'stack')
          # })
          ######


          ###### Sales :
```

```
          output$dwdSales <- downloadHandler(
            filename = function() {
              paste("SalesData", Sys.Date(), input$typeSales, sep=".")
            },
            content = function(file) {
              if(input$typeSales == "xlsx") {
                writexl::write_xlsx(sales[sales$SubmitterID %in% sub
                                          & sales$Year >= 2016
                                          & sales$Year <= as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]), ], file)
              } else {
                write.csv(sales[sales$SubmitterID %in% sub
                                & sales$Year >= 2016
                                & sales$Year <= as.integer(stringr::str_split(string = Sys.Date(),
pattern = "-")[[1]][1]), ],
                          file= file,
                          row.names=F)
              }
            }
          )

          output$salesData <- plotly::renderPlotly({

            if( all(sub %in% unique(sales$SubmitterID)) && length(sub) < 2){
              tmp <- sales[sales$SubmitterID %in% sub, c("Year", "SalesVolume")]

              # Filter and Clean Data
              tmp <- dplyr::filter(tmp, tmp$Year >= 2016)
              tmp <- dplyr::filter(tmp, tmp$Year <= as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]))

              if(!is.null(tmp)) {
                df <- stats::aggregate(.~Year, data = tmp, FUN = sum)
                fig <- plotly::plot_ly(df, x = ~Year, y = ~SalesVolume, name = 'SalesVolume', type =
'bar')
              } else {
                return(NULL)
              }

            } else {
              return(NULL)
            }

          })
        }

        # If no submitter selected :
################################################################################
        else {
          observeEvent(input$choice,{

            if(input$choice == "All"){

              output$regionChoice <- NULL
              output$countryChoice <- NULL
              updateSelectizeInput(session = session, inputId = 'submitterId', choices =
submitters$submitterIdName, server = TRUE)

              output$boxSubmitters <- shinydashboard::renderInfoBox({
                valueBox(
                  value = length(submitters$SubmitterID),
                  subtitle = "Number of submitters",
                  width = 2,
                  color = "purple",
                  icon = icon("user")
                )
              })

              output$boxProduct <- shinydashboard::renderInfoBox({
                valueBox(
                  value = length(product$ProductID),
                  subtitle = "Number of products",
                  width = 2,
                  color = "blue",
                  icon = icon("cube")
                )
              })
```

```
                output$boxPresentation <- shinydashboard::renderInfoBox({
                  valueBox(
                    value = length(presentation$SubmitterID),
                    subtitle = "Number of presentations",
                    width = 2,
                    color = "light-blue",
                    icon = icon("cubes")
                  )
                })


                ###### Product repartition :
                output$dwdpieChart <- downloadHandler(
                  filename = function() {
                    paste("ProductType", Sys.Date(), input$type, sep=".")
                  },
                  content = function(file) {
                    if(input$type == "xlsx") {
                      writexl::write_xlsx(product, file)
                    } else {
                      write.csv(product,
                                file= file,
                                row.names=F)
                    }
                  }
                )

                output$pieChart <- plotly::renderPlotly({
                  tp <- as.data.frame(table(product$ProductType))
                  colnames(tp) <- c("Type", "Quantity")
                  tp$Type <- RefTobaccoProductType[RefTobaccoProductType$item %in% tp$Type,]$label
                  productType <- tp$Type
                  qtt <- tp$Quantity

                  fig <- plot_ly(tp, labels = ~Type, values = ~qtt, type = 'pie',
                                 marker=list(colors = palette[tp$Type]))
                  fig <- fig %>% layout(title = 'Global repartition of product type',
                                        xaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE),
                                        yaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE))
                })
                ##########

                ########## BrandNames :
                output$dwdBrandname <- downloadHandler(
                  filename = function() {
                    paste("BrandNames", Sys.Date(), input$typeBrandname, sep=".")
                  },
                  content = function(file) {
                    if(input$typeBrandname == "xlsx") {
                      writexl::write_xlsx(presentation[ , c("ProductID", "LaunchDate",
"BrandName","BrandSubtypeName")][input$brandname_rows_all,], file)
                    } else {
                      write.csv(presentation[ , c("ProductID", "LaunchDate",
"BrandName","BrandSubtypeName")][input$brandname_rows_all,],
                                file= file,
                                row.names=F)
                    }
                  }
                )

                # if(input$type == "xlsx") {
                #   writexl::write_xlsx(dataBase[[input$tables]][input$dataFrame_rows_all,], file)
                # } else {
                #   write.csv(dataBase[[input$tables]][input$dataFrame_rows_all,],
                #             file= file,
                #             row.names=F)
                # }

                output$brandname <- DT::renderDT({
                  brandNameDf <- presentation[order(as.Date(presentation$LaunchDate, format =
"%Y%m/%d")),]
                  DT::datatable(
                    brandNameDf[, c("ProductID", "LaunchDate","BrandName","BrandSubtypeName")],
                    rownames = FALSE,
                    selection = "none",
                    filter = 'top'
```

```
    )
  })
  #########


  ######### Nicotine :
  # output$dwdNicotine <- downloadHandler(
  #   filename = function() {
  #     paste("nicotineConcentration", Sys.Date(), input$typeNicotine, sep=".")
  #   },
  #   content = function(file) {
  #     if(input$typeNicotine == "xlsx") {
  #       writexl::write_xlsx(product[product$Nicotine <= 20, ], file)
  #     } else {
  #       write.csv(product[product$Nicotine <= 20, ],
  #                 file= file,
  #                 row.names=F)
  #
  #     }
  #   }
  # )
  #
  # output$nicotine <- plotly::renderPlotly({
  #
  #   tp <- as.data.frame(table(round(product$Nicotine)))
  #   colnames(tp) <- c("Concentration", "Number of product")
  #   tp$Concentration <- as.numeric(tp$Concentration)
  #
  #   tp <- tp[tp$Concentration <= 20,]
  #   tp$Concentration <- as.factor(tp$Concentration)
  #
  #   concentration <- tp$Concentration
  #   qtt <- tp$`Number of product`
  #
  #   tp %>%
  #     mutate(id = as.numeric(1)) %>%
  #     plot_ly(x = ~id, y = ~qtt, color = ~concentration, colors =
paletteNico[concentration], type = 'bar') %>%
  #     layout(yaxis = list(title = 'qtt'), barmode = 'stack')
  # })
  #########

  ######### Sales :
  output$dwdSales <- downloadHandler(
    filename = function() {
      paste("SalesData", Sys.Date(), input$typeSales, sep=".")
    },
    content = function(file) {
      if(input$typeSales == "xlsx") {
        writexl::write_xlsx(sales[sales$Year >= 2016
                                  & sales$Year <= as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]), ], file)
      } else {
        write.csv(sales[sales$Year >= 2016
                        & sales$Year <= as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]), ],
                  file= file,
                  row.names=F)

      }
    }
  )

  output$salesData <- plotly::renderPlotly({

    # Filter and Clean Data
    sales <- dplyr::filter(sales, sales$Year>=2016)
    sales <- dplyr::filter(sales, sales$Year<=as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]))

    sales <- sales[ ,c("Year", "SalesVolume")]
    df <- stats::aggregate(.~Year, data = sales, FUN = sum)
    fig <- plotly::plot_ly(df, x = ~Year, y = ~SalesVolume, name = 'SalesVolume', type =
'bar')
  })

} else if(input$choice == "By Region"){
  output$countryChoice <- NULL
```

```
              output$regionChoice <- renderUI({
                selectizeInput(inputId = ns("regionChoice"), label = "Select the region", choices =
unique(submitters$region), selected = "Europe", multiple = TRUE,
                               options = list(
                                 placeholder = "Select a region",
                                 'plugins' = list('remove_button'),
                                 # 'create' = TRUE,
                                 'persist' = FALSE
                               ))
              })

              ################################################################################

              observeEvent(input$regionChoice, {

                sub_df <- submitters[submitters$region %in% input$regionChoice,]
                updateSelectizeInput(session = session, inputId = 'submitterId', choices =
submitters[submitters$region %in% input$regionChoice,]$submitterIdName, server = TRUE)

                output$boxSubmitters <- shinydashboard::renderInfoBox({
                  valueBox(
                    value = length(sub_df$SubmitterID),
                    subtitle = "Number of submitters",
                    width = 2,
                    color = "purple",
                    icon = icon("user")
                  )
                })

                output$boxProduct <- shinydashboard::renderInfoBox({
                  valueBox(
                    value = length(product[product$SubmitterID %in% sub_df$SubmitterID,]$ProductID),
                    subtitle = "Number of products",
                    width = 2,
                    color = "blue",
                    icon = icon("cube")
                  )
                })

                output$boxPresentation <- shinydashboard::renderInfoBox({
                  valueBox(
                    value = length(presentation[presentation$SubmitterID %in% sub_df$SubmitterID,
]$SubmitterID),
                    subtitle = "Number of presentations",
                    width = 2,
                    color = "light-blue",
                    icon = icon("cubes")
                  )
                })

                output$pieChart <- NULL

                req(input$regionChoice)

                ###### ZONE DE TEST
                output$dwdpieChart <- downloadHandler(
                  filename = function() {
                    paste("ProductType", Sys.Date(), input$type, sep=".")
                  },
                  content = function(file) {
                    if(input$type == "xlsx") {
                      writexl::write_xlsx(product[product$SubmitterID %in% sub_df$SubmitterID,],
file)
                    } else {
                      write.csv(product[product$SubmitterID %in% sub_df$SubmitterID,],
                                file= file,
                                row.names=F)
                    }
                  }
                )


                output$pieChart <- plotly::renderPlotly({
                  tp <- as.data.frame(table(product[product$SubmitterID %in%
sub_df$SubmitterID,]$ProductType))
                  colnames(tp) <- c("Type", "Quantity")
                  tp$Type <- RefTobaccoProductType[RefTobaccoProductType$item %in% tp$Type,]$label
                  productType <- tp$Type
```

```
                qtt <- tp$Quantity
                fig <- plot_ly(tp, labels = ~Type, values = ~qtt, type = 'pie',
                               marker=list(colors = palette[tp$Type]))
                fig <- fig %>% layout(title = paste('Repartition of product type',
input$regionChoice, sep = " - "),
                                      xaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE),
                                      yaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE))
              })
              ##########

              ########## Brand Names :
              output$dwdBrandname <- downloadHandler(
                filename = function() {
                  paste("BrandNames", Sys.Date(), input$typeBrandname, sep=".")
                },
                content = function(file) {
                  if(input$typeBrandname == "xlsx") {
                    writexl::write_xlsx(presentation[presentation$SubmitterID %in%
sub_df$SubmitterID, c("ProductID", "LaunchDate",
"BrandName","BrandSubtypeName")][input$brandname_rows_all,], file)
                  } else {
                    write.csv(presentation[presentation$SubmitterID %in% sub_df$SubmitterID,
c("ProductID", "LaunchDate", "BrandName","BrandSubtypeName")][input$brandname_rows_all,],
                              file= file,
                              row.names=F)
                  }
                }
              )
              output$brandname <- DT::renderDT({
                brandNameDf <- presentation[order(as.Date(presentation$LaunchDate, format =
"%Y%m/%d")),]
                DT::datatable(
                  brandNameDf[brandNameDf$SubmitterID %in% sub_df$SubmitterID, c("ProductID",
"LaunchDate","BrandName","BrandSubtypeName")],
                  rownames = FALSE,
                  selection = "none",
                  filter = 'top'
                )
              })
              ##########

              ########## Nicotine :
              # output$dwdNicotine <- downloadHandler(
              #   filename = function() {
              #     paste("nicotineConcentration", Sys.Date(), input$typeNicotine, sep=".")
              #   },
              #   content = function(file) {
              #     if(input$typeNicotine == "xlsx") {
              #       writexl::write_xlsx(product[product$SubmitterID %in% sub_df$SubmitterID
              #                                  & product$Nicotine <= 20, ], file)
              #     } else {
              #       write.csv(product[product$SubmitterID %in% sub_df$SubmitterID
              #                         & product$Nicotine <= 20, ],
              #                 file= file,
              #                 row.names=F)
              #     }
              #   }
              # )
              #
              # output$nicotine <- plotly::renderPlotly({
              #
              #   tp <- as.data.frame(table(round(product[product$SubmitterID %in%
sub_df$SubmitterID,]$Nicotine)))
              #   colnames(tp) <- c("Concentration", "Number of product")
              #   tp$Concentration <- as.numeric(tp$Concentration)
              #
              #   tp <- tp[tp$Concentration <= 20,]
              #   tp$Concentration <- as.factor(tp$Concentration)
              #
              #   concentration <- tp$Concentration
              #   qtt <- tp$`Number of product`
              #
              #   tp %>%
              #     mutate(id = as.numeric(1)) %>%
              #     plot_ly(x = ~id, y = ~qtt, color = ~concentration, colors =
paletteNico[concentration], type = 'bar') %>%
```

```
                #     layout(yaxis = list(title = 'qtt'), barmode = 'stack')
                # })

                ##########

                ########## Sales :
                output$dwdSales <- downloadHandler(
                  filename = function() {
                    paste("SalesData", Sys.Date(), input$typeSales, sep=".")
                  },
                  content = function(file) {
                    if(input$typeSales == "xlsx") {
                      writexl::write_xlsx(sales[sales$SubmitterID %in% sub_df$SubmitterID
                                                & sales$Year >= 2016
                                                & sales$Year <= as.integer(stringr::str_split(string
= Sys.Date(), pattern = "-")[[1]][1]), ], file)
                    } else {
                      write.csv(sales[sales$SubmitterID %in% sub_df$SubmitterID
                                      & sales$Year >= 2016
                                      & sales$Year <= as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]), ],
                                file= file,
                                row.names=F)
                    }
                  }
                )
                output$salesData <- plotly::renderPlotly({

                  sales <- sales[sales$SubmitterID %in% sub_df$SubmitterID, c("Year",
"SalesVolume")]
                  # Filter and Clean Data
                  sales <- dplyr::filter(sales, sales$Year>=2016)
                  sales <- dplyr::filter(sales, sales$Year<=as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]))

                  df <- stats::aggregate(.~Year, data = sales, FUN = sum)
                  fig <- plotly::plot_ly(df, x = ~Year, y = ~SalesVolume, name = 'SalesVolume', type
= 'bar')
                })

              }, ignoreNULL = FALSE)

            } else if(input$choice == "By Country"){

              output$regionChoice <- NULL
              output$countryChoice <- renderUI({

                # Get flags :
                img_urls <- paste0(
                  "https://cdn.rawgit.com/lipis/flag-icon-css/master/flags/4x3/",
                  tolower(unique(submitters$Country)), ".svg"
                )

                multiInput(
                  inputId = ns("pays"),
                  label = "Select the country : ",
                  choices = NULL,
                  selected = "France",
                  choiceNames = lapply(
                    seq_along(unique(submitters$countriesNames)),
                    function(i) {
                      tagList(
                        tags$img(src = img_urls[i], width = 20, height = 15),
                        unique(submitters$countriesNames)[i]
                      )
                    }
                  ),
                  choiceValues = unique(submitters$countriesNames)
                )
              })


              observeEvent(input$pays, {

                sub_df <- submitters[submitters$countriesNames %in% input$pays,]

                updateSelectizeInput(session = session, inputId = 'submitterId', choices =
submitters[submitters$countriesNames %in% input$pays,]$submitterIdName, server = TRUE)
```

```
                output$boxSubmitters <- shinydashboard::renderInfoBox({
                  valueBox(
                    value = length(sub_df$SubmitterID),
                    subtitle = "Number of submitters",
                    width = 2,
                    color = "purple",
                    icon = icon("user")
                  )
                })

                output$boxProduct <- shinydashboard::renderInfoBox({
                  valueBox(
                    value = length(product[product$SubmitterID %in% sub_df$SubmitterID,]$ProductID),
                    subtitle = "Number of products",
                    width = 2,
                    color = "blue",
                    icon = icon("cube")
                  )
                })

                output$boxPresentation <- shinydashboard::renderInfoBox({
                  valueBox(
                    value = length(presentation[presentation$SubmitterID %in% sub_df$SubmitterID,
]$SubmitterID),
                    subtitle = "Number of presentations",
                    width = 2,
                    color = "light-blue",
                    icon = icon("cubes")
                  )
                })

                output$pieChart <- NULL
                req(input$pays)

                ###### ZONE DE TEST
                output$dwdpieChart <- downloadHandler(
                  filename = function() {
                    paste("ProductType", Sys.Date(), input$type, sep=".")
                  },
                  content = function(file) {
                    if(input$type == "xlsx") {
                      writexl::write_xlsx(product[product$SubmitterID %in% sub_df$SubmitterID,],
file)
                    } else {
                      write.csv(product[product$SubmitterID %in% sub_df$SubmitterID,],
                                file= file,
                                row.names=F)
                    }
                  }
                )

                output$pieChart <- plotly::renderPlotly({
                  tp <- as.data.frame(table(product[product$SubmitterID %in%
sub_df$SubmitterID,]$ProductType))
                  colnames(tp) <- c("Type", "Quantity")
                  tp$Type <- RefTobaccoProductType[RefTobaccoProductType$item %in% tp$Type,]$label
                  productType <- tp$Type
                  qtt <- tp$Quantity
                  fig <- plot_ly(tp, labels = ~Type, values = ~qtt, type = 'pie',
                                 marker=list(colors = palette[tp$Type]))
                  fig <- fig %>% layout(title = paste('Repartition of product type', input$pays, sep
= " - "),
                                        xaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE),
                                        yaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE))
                })
                ##########

                ########## BrandName :
                output$dwdBrandname <- downloadHandler(
                  filename = function() {
                    paste("BrandNames", Sys.Date(), input$typeBrandname, sep=".")
                  },
                  content = function(file) {
                    if(input$typeBrandname == "xlsx") {
```

```
                    writexl::write_xlsx(presentation[presentation$SubmitterID %in%
sub_df$SubmitterID , c("ProductID", "LaunchDate",
"BrandName","BrandSubtypeName")][input$brandname_rows_all,], file)
                  } else {
                    write.csv(presentation[presentation$SubmitterID %in% sub_df$SubmitterID ,
c("ProductID", "LaunchDate", "BrandName","BrandSubtypeName")][input$brandname_rows_all,],
                              file= file,
                              row.names=F)
                  }
                }
              )

              output$brandname <- DT::renderDT({
                brandNameDf <- presentation[order(as.Date(presentation$LaunchDate, format =
"%Y%m/%d")),]
                DT::datatable(
                  brandNameDf[brandNameDf$SubmitterID %in% sub_df$SubmitterID, c("ProductID",
"LaunchDate", "BrandName","BrandSubtypeName")],
                  rownames = FALSE,
                  selection = "none",
                  filter = 'top'
                )
              })
              ##########

              ########## Nicotine :
              # output$dwdNicotine <- downloadHandler(
              #   filename = function() {
              #     paste("nicotineConcentration", Sys.Date(), input$typeNicotine, sep=".")
              #   },
              #   content = function(file) {
              #     if(input$typeNicotine == "xlsx") {
              #       writexl::write_xlsx(product[product$SubmitterID %in% sub_df$SubmitterID
              #                                  & product$Nicotine <= 20, ], file)
              #     } else {
              #       write.csv(product[product$SubmitterID %in% sub_df$SubmitterID
              #                         & product$Nicotine <= 20, ],
              #                 file= file,
              #                 row.names=F)
              #     }
              #   }
              # )
              # output$nicotine <- plotly::renderPlotly({
              #
              #   tp <- as.data.frame(table(round(product[product$SubmitterID %in%
sub_df$SubmitterID,]$Nicotine)))
              #   colnames(tp) <- c("Concentration", "Number of product")
              #   tp$Concentration <- as.numeric(tp$Concentration)
              #
              #   tp <- tp[tp$Concentration <= 20,]
              #   tp$Concentration <- as.factor(tp$Concentration)
              #
              #   concentration <- tp$Concentration
              #   qtt <- tp$`Number of product`
              #
              #   tp %>%
              #     mutate(id = as.numeric(1)) %>%
              #     plot_ly(x = ~id, y = ~qtt, color = ~concentration, colors =
paletteNico[concentration], type = 'bar') %>%
              #     layout(yaxis = list(title = 'qtt'), barmode = 'stack')
              # })
              #############

              ############# Sales :
              output$dwdSales <- downloadHandler(
                filename = function() {
                  paste("SalesData", Sys.Date(), input$typeSales, sep=".")
                },
                content = function(file) {
                  if(input$typeSales == "xlsx") {
                    writexl::write_xlsx(sales[sales$SubmitterID %in% sub_df$SubmitterID
                                              & sales$Year >= 2016
                                              & sales$Year <= as.integer(stringr::str_split(string
= Sys.Date(), pattern = "-")[[1]][1]), ], file)
                  } else {
                    write.csv(sales[sales$SubmitterID %in% sub_df$SubmitterID
                                    & sales$Year >= 2016
```

```
                                    & sales$Year <= as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]), ],
                                  file= file,
                                  row.names=F)
                    }
                  }
                )

                output$salesData <- plotly::renderPlotly({

                  sales <- sales[sales$SubmitterID %in% sub_df$SubmitterID, c("Year",
"SalesVolume")]
                  # Filter and Clean Data
                  sales <- dplyr::filter(sales, sales$Year>=2016)
                  sales <- dplyr::filter(sales, sales$Year<=as.integer(stringr::str_split(string =
Sys.Date(), pattern = "-")[[1]][1]))

                  df <- stats::aggregate(.~Year, data = sales, FUN = sum)
                  fig <- plotly::plot_ly(df, x = ~Year, y = ~SalesVolume, name = 'SalesVolume', type
= 'bar')
                })
              }, ignoreNULL = FALSE)
            } ## end last condition
          })# end observe event
        } # end else
      }, ignoreNULL = FALSE) # end observe envent submitter id

      t2 <- Sys.time()
      print("Temps submitter : ")
      print(t2 - t1)
    }) #end observe
  }) # end module
} # end server

## To be copied in the UI
# mod_submitters_ui("submitters_1")

## To be copied in the server
# mod_submitters_server("submitters_1", r=r)
```