Data Quality Dashboard Development Guide

## Introduction

This document serves as a comprehensive guide aiming to allow anyone to expand on the Data Quality Dashboard, linked here: [link](https://scotland.shinyapps.io/phs-dqa/).  
It provides the basic guidelines on how to add more tabs, edit tables, add text and more.

## The file structure of the dashboard

It is crucial to understand the structure of the files behind the dashboard before editing it.  
The dashboard is split into a number of folders, with two of them being the most important - ‘app’ and ‘code’.

├── dqa\_dashboard.Rproj  
├── README.md  
├── app  
├── code  
│   ├── clinical\_coding\_tab  
│   ├── data\_quality\_tab  
│   ├── home\_tab  
├── data  
└── functions

### The ‘app’ folder

The ‘app’ folder contains three files - server.R, ui.R and global.R.

The function of the global.R file is to read in all the files and libraries, as well as to conduct some of the processes that cannot easily be conducted externally, such as rendering maps. Think of global.R as a corridor with a storage room at the end of it. It connects the functional rooms such as kitchen and living room and stores appliances needed by both, such as bedding or kitchen equipment.

The server.R file creates all the outputs that will later appear in the dashboard. The code written here defines each of the elements appearing later in the dashboard, that is rendering tables, maps and graphs. Think of server.R as a kitchen, where you prepare all the dishes but you actually serve them to your guests in the living room.

The ui.R file decides the structure of the dashboard. It takes all the elements rendered in the server.R and orders them in the correct tabs (which are also defined in the ui.R file) and wraps text around them. It is analogous to ordering images of tables in a word document. You created them in Excel, but now need to produce a cohesive report, with the tables in the right sections wrapped in explanations. ui.R is therefore the living room, where you present all your dishes to the guests on fancy plates.

### The ‘code’ folder

The ‘code’ folder was created to allow for an easy way to distinguish different data sections. Each of the main tabs has its own sub-folder and if you were to add your own tab, it is advised to create a new sub-folder in the ‘code’ folder and name it accordingly. In the tab sub-folder, you will find files for each of the sections of the tab. For instance, the ‘data\_quality\_tab’ contains three files - data\_monitoring\_timeliness.R, data\_monitoring\_completeness.R and smr\_audit.R, each of them using a different dataset. Those files are used to handle data - to get it from the SQL server, clean and organise it. After all the data wrangling, you simply write it into an excel file so that it can be passed on to the dashboard. Thanks to the fact that data is processed externally from the dashboard and then it’s exported, the dashboard itself can run much faster as it does not waste resources on processing the data each time it loads. We will delve deeper into how to write and read your data in the next section.

## Adding your own data

**Before executing any code, load the libraries at the top of the current file - select all lines starting with ‘library’ and press ctrl+Enter.**  
You will notice that I did not discuss the other folders, particularly ‘data’. This is where you save all your processed data files. If it so happens that you have your data already prepared in the form you want to show it on the dashboard, you can just copy it directly into that folder. However, if you needed to process it first in the ‘code’ folder, at the end of the file you worked in, type the following code:

write\_csv(table\_name, here::here("data", "file\_name.csv"))

This will save the table you created in R (table\_name) as ‘file\_name.csv’ in the folder ‘data’. The here::here part is a useful shortcut allowing to unify the directories across different dashboard developers.

Following that command you will see the file appear in the ‘data’ folder.

You can now go to global.R and read in the file you created:

object\_name <- read\_csv(here::here("data", "file\_name.csv"))

To do that, you used the read\_csv function. You need to define the data as a variable (in this case object\_name). Use an intuitive name that explains well what the data represents as you will use it throughout the dashboard from now on. The function is asked to search in the folder ‘data’ (first input) for a file named ‘file\_name.csv’ (second input). Good job, you can now use your data in the dashboard. If you want to change some of it, go back to the file you wrote it in, make the changes, write and read it in again.

## Create a new tab

Before you add your table to the dashboard, create a tab to place it in. To do that, first go to ui.R. You will see a large portion of code encompassing all of the dashboard’s layout. Currently there are three main tabs at the top - ‘Home’, ‘Data Quality’ and ‘Coding Discrepancies and Issues’. If you look at the structure of the [ui.R file](https://github.com/Public-Health-Scotland/dqa_dashboard/blob/main/app/ui.R), you will see a lot of lines starting with tabPanel.

TabPanel is responsible for creating the panels at the top, but also those within each tab. For instance, in the ‘Data Quality’ tab, there are four sub-tabs: ‘Home’, ‘Completeness’, ‘Timeliness’ and ‘Accuracy Scores from SMR Audits’. As you can see they all start with tabPanel, but the sub-tabs are indented further to the right, making it easier to recognise at which level they are embedded within the dashboard. To create a new tab at the top, you want to add a tabPanel at the same level as e.g. ‘Coding Discrepancies and Issues’. Tabs are added from left to right in the same order as listed in the ui.R, so if you want to add a new tab to the right of the right-most one, put it at the end of ui.R. Add the following code where in the appropriate position:

,  
 tabPanel(  
 title = 'Dashboard Test'  
 )

You will need to start with a comma and pay attention to the round brackets as these can get tricky when there are too many tabs. If in doubt which brackets you should wrap around your code and which to leave behind it, try double-clicking at one of the brackets. It will automatically highlight all of the code encompassed by the bracket you selected. You want the above tab to be outside of the region of the previous tab - ‘Coding Discrepancies and Issues’. This means that the correct code should look like this:

[...]  
 tabPanel("SMR01 ICD-10 Symptom R Codes",  
 selectInput('yearR', 'Choose year:',  
 choices = c('(All)', sort(unique(RCodes\_table$year)))  
 ),  
 DT::dataTableOutput("RCodes")  
 )  
 )),  
tabPanel(  
 title = 'Dashboard Test'  
)  
[...]

Now that you’ve created the tab, let’s render the table so that you can place it within the dashboard. Recall that you’ve read in your hypothetical table already and saved it as ‘object\_name’. Now go to server.R. You will notice that the structure of this file is wrapped within the following code:

shinyServer(function(input, output, session) {  
   
})

The order of code chunks within server.R is irrelevant from a technical perspective and only matters when considering legibility. You can now render the data table as follows. We will also add a host of additional functionalities, such as filters or the ability to download the table. Each of these will be commented on in the code chunk:

shinyServer(function(input, output, session) {  
   
 output$table\_test <- DT::renderDataTable({ #table\_test is the name you give to the table which will later be called in ui.R  
 test\_filter <- object\_name %>% #create a filter and start working on the table you created earlier as object\_name  
 filter(year == input$year)%>% #let's say you want to filter your data for year, input$year will correspond to a dropdown in the UI  
 rename("Healthboard"="HBName", #this is a useful method to rename any columns that might have inconvenient names in the original data where the first variable is the desired column name and the second is the current name  
 "Year"="year", "Percentage"="percentage\_1")  
   
 dtable\_error1 <- datatable(data = test\_filter, #call the data that uses the filter (test\_filter)   
 escape = FALSE,  
 rownames = FALSE, #do not show rownames  
 class="compact stripe hover",  
 selection = 'none',  
 extension = 'Buttons', #this is necessary to create the download buttons  
 options = list(  
 rowsGroup = list(0),  
 columnDefs = list(  
 list(className = 'dt-center', targets = "\_all")  
 ),  
 pageLength = 15, #here you can set the default number of rows displayed  
 dom = 'Bfrtip', #used to accommodate the download buttons  
 buttons = c('copy', 'csv', 'excel', 'pdf') #list download buttons that you'd like to include  
 )  
   
 )  
 })  
})

The code above will allow you to create quite an advanced table that is also customisable. However, it will not yet display in the dashboard as it is not in the ui.R. Go back to the tabPanel you created and insert the following code inside:

tabPanel(  
 title = 'Dashboard Test',  
 selectInput('year', 'Choose year:',   
 choices = unique(object\_name$year)), #here using the 'unique' function to create a dropdown menu of all available years in the data  
 DT::dataTableOutput("table\_test")) #use the same name given to the table in the server so that the dashboard knows which objects to link  
 )

And that’s it, your table should now display properly in the tabPanel. You can play around with different tab layouts. There are also other options such as menus and sidebars. You can view the full list of ui navigation options here: [link](https://shiny.rstudio.com/articles/layout-guide.html).

You can also render graphs and charts if you create them in ggplot and then export them. You can read more about plots in RShiny here: [link](https://shiny.rstudio.com/articles/plot-interaction.html).

## Using the lookups

It often happens that you pull your data using a SQL Query from SMRA and you only have codes for the healthboards but don’t have their names. Here is a simple fix using a file in the ‘lookups’ folder assuming you called the data you pulled ‘test\_data’ and the column containing healthboard codes is called “hbres\_currentdate”. Lookup is a file containing both the healthboard codes and names so that it can be joined with the original dataset by the codes which are present in both datasets, but it will result in the healthboard names being included in the original dataset as well:

#Read in hb\_lookup file:  
hb\_lookup <- read\_csv(here::here("lookups", "hb\_lookup.csv")) #load in the lookup file  
  
#append hb names to diagnosis2 dataframe  
test\_data <- left\_join(test\_data, hb\_lookup[c(1:2)], by = c("hbres\_currentdate"="HB")) #join the original dataset with the lookup dataset  
test\_data #display the test\_data

## Other useful guides

There are two well curated guides that are of use for this dashboard - one teaches collaboration via gitHub, the other building maps based on Scottish healthboards. Both are available through TPP.