Case Study - Operational risk management tool

App.R

The application described in this case study has been uploaded to a GitHub repository <github – link here> and provides an example for an operational risk management database and workflow tool used in risk management across different industries. Operational risk is defined as losses or potential losses caused by failed processes, systems, human errors as well as external factors. One of the key techniques in operational risk is the efficient and effective collection, classification and management of internal loss data. As such, data is key here and with that the need for capturing this data across the organization. While there are very powerful solutions out there that might cover the needs of large corporates and large financial institutions, the case study provided here is more focusing on small and medium-sized enterprises covering their risk management needs.

The application provided in the case study focuses on several aspects of operational risk management:

1. Workflow tool: registered users use a report with predefined fields to submit an operational incident. The incident immediately gets stored in the database and, in parallel, the tool notifies the operational risk manager about the new case.
2. Database: all incidents, comments, discussions around an operational incident get stored in the database.
3. Dashboard: once reporting into the tool gains momentum, the built-in dashboard can be used to present a good summary of the current data as well as help defined Key Risk Indicators (KRIs) in the context of operational risk.
4. Tracking/ Audibility: since every submission to the database is stored, it is easy to download or extract information on operational incidents.
5. Communication: the application and the data collected therein supports data-based communication vis-à-vis senior executives and can be used as a basis for discussions during committees.
6. Education and training: we have enriched the application with links to useful documents such as training materials as well as comments on the specific fields so that the user can access information within the same environment.
7. Documentation: good governance is the basis for an efficient risk management program. As such, the application in the case study also gives access to the current version of the policy or any other governance-related document covering the topic of operational risk.

In addition to the above-mentioned key aspects covered, several built-in functions of the tool are worth mentioning since they support good governance:

1. Login: only registered users can access the application (see section on login.R). An administrator, e.g. operational risk manager, can set up new users and make sure those users are properly trained before submitting cases through the tool.
2. Role-based approach: the user management presented in the case study has several fields that allow for a role-based steering of user access.
   1. Region/ legal entity: the flags “Region” and “Entity” can for example be used to restrict access to users of a particular region only seeing cases from that region (this is the way how it is coded in the current version of the tool).
   2. Read/ Write: several tabs, like the “Update” tab, can only be accessed based on a particular role which is managed in the flag “User\_right” (R or W)
   3. Risk Owner: the flag “Risk\_Owner” can be used to define users to who incidents can be assigned.
3. E-mail notification: the connection to the SMTP server (as described in chapter 2) is a key element here. In the case study, the e-mail notification has only been defined for the initial submission of an operational incident. However, the functionality can be extended into covering every update, notifying risk owners, providing standardized reports and many more.

The application used in the case study offers several options to handle data:

* Data input
  + Form used to structurally input data into the database. In the case study presented, an .sqlite database has been used to store data.
* Data output:
  + Dashboard function is part of the case study presented here, displaying data on the screen in the form of
    - Datatables (using package DT), allowing for a filtering of data
    - Value boxes (package shinydashboard), allowing for a summary and KRI representation of data
    - Charts (e.g. piechart, wordcloud,…)
  + Data is being stored in a database (.sqlite format)
  + Data can be directly downloaded by the user of the tool (in the case study, a .csv format has been used)
  + Finally, data can be shared per e-mail (SMTP) as an Excel file (see section above)

Another alternative would be a .pdf, .docx or .html report compiled (for e.g. senior executives) using the packages knitr and rmarkdown.

Login.R

The login.R script uses modalDialog functions and acts as a gatekeeper to the respective application. This should be included as part of every application that has been rolled out internally, making sure that only registered users are accessing the applications.

Access to the application is managed through entries into a database stored in the data folder. The login.R example below uses a username and password combination stored in a .sqlite file managed by the administrator of the application.

login.R

vals <- reactiveValues(UserType = "User")

# Return the UI for a modal dialog with data selection input. If 'failed' is

# TRUE, then display a message that the previous value was invalid.

dataModal\_login <- function(failed = FALSE) {

modalDialog(

textInput("UserId", "UserId:"),

passwordInput("password", "Password:"),

if (failed)

div(tags$b("Invalid username or password", style = "color: red;")),

footer = tagList(

actionButton("ok", "Login"),

br(),

br(),

actionLink("register","Change password"),

br(),

br(),

actionLink("ooops","Go back")

)

)

}

#

data2Modal <- function(failed = FALSE) {

modalDialog(

textInput("UserId2", "UserId:"),

passwordInput("password\_old", "Old password:"),

passwordInput("password\_new", "New password:"),

if (failed)

div(tags$b("Invalid username or password", style = "color: red;")),

footer = tagList(

actionButton("okay", "OK")

)

)

}

#observeEvent(input$logout, {

# session$reload()

#})

# Show modal when button is clicked.

observe({

# if (input$tabs == "intro") {

showModal(dataModal\_login())

#}

})

# When OK button is pressed, attempt to load the data set. If successful,

# remove the modal. If not show another modal, but this time with a failure

# message.

observeEvent(input$ok, {

con <- dbConnect(SQLite(), dbname = "data/test.sqlite")

Users <- dbGetQuery(con, "SELECT \* FROM tab")

dbDisconnect(con)

# Check that data object exists and is data frame.

if (input$UserId %in% Users$UserId && input$password == Users$password[which(Users$UserId == input$UserId)]) {

vals$UserType <- Users$Type[which(Users$UserId == input$UserId)]

removeModal()

} else {

showModal(dataModal\_login(failed = TRUE))

}

})

observeEvent (input$register,{

showModal(data2Modal())

})

observeEvent (input$ooops,{

updateTabItems(session, "tabs", selected = "intro")

})

observeEvent(input$okay,{

#con <- dbConnect(SQLite(), dbname = "Users.sqlite")

#Users <- dbGetQuery(con, "SELECT \* FROM tab")

#dbDisconnect(con)

con <- dbConnect(SQLite(), dbname = "data/test.sqlite")

Users <- dbGetQuery(con, "SELECT \* FROM tab")

dbDisconnect(con)

if (input$UserId2 %in% Users$UserId && input$password\_old == Users$password[which(Users$UserId == input$UserId2)]) {

con <- dbConnect(SQLite(), dbname = "data/test.sqlite")

ff <- sprintf("UPDATE tab SET '%s' = '%s' WHERE UserId = '%s'", "password", input$password\_new, input$UserId2)

dbGetQuery(con, ff)

dbDisconnect(con)

vals$UserType <- Users$Type[which(Users$UserId == input$UserId2)]

removeModal()

} else {

showModal(data2Modal(failed = TRUE))

}

})