```r
### SCRIPT 5 - R ###
# Obtención de plazos de ejecucion desde ATOM
#
#########################################################
library(XML)
library(dplyr)
library(foreach)
library(doParallel)

# Configuración paralela
cores <- detectCores()
cl <- makeCluster(cores[1] - 1, type = "PSOCK", outfile = "log.txt")
registerDoParallel(cl)

# Ruta de trabajo
setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP")
PATH <- "licitacionesPerfilesContratanteCompleto3_2025"  # carpeta donde están los .atom
RData_FILE <- "3.34_PlannedPeriod_Extension_2025.Rdata"
DBtablename <- "PlannedPeriodExtension"

# Función para encontrar un nodo padre por nombre
DamePadre <- function(xml_a, tag) {
  xml_p <- xml_a
  repeat {
    xml_p <- xmlParent(xml_p)
    if (xmlName(xml_p) == tag) break
  }
  return(xml_p)
}

# Función segura para extraer el valor de un nodo XML
safe_xml_value <- function(node, tag) {
  if (!is.null(node) && !is.null(node[[tag]])) {
    return(xmlValue(node[[tag]]))
  } else {
    return(NA)
  }
}

# Columnas objetivo
nombres <- c("ContractID", "entryID",
             "start_date", "end_date",
             "duration", "duration_unit",
             "description", "extension_description",
             "updated")

# Inicializar data.frame vacío
Proy <- data.frame(matrix(NA, 0, length(nombres)))
colnames(Proy) <- nombres

# Listar archivos atom
arch <- list.files(path = PATH, pattern = "\\.atom$", full.names = FALSE)

# Procesar en paralelo
Proy <- foreach(ci = 1:length(arch), .combine = rbind, .packages = c("XML", "dplyr")) %dopar% {

  archivo_temp <- xmlParse(file.path(PATH, arch[ci]))
  cat(sprintf("[%s] Procesando archivo %s\n", Sys.time(), arch[ci]))

  buscar <- xmlElementsByTagName(xmlRoot(archivo_temp), "ProcurementProject", recursive = TRUE)

  Proy <- data.frame(matrix(NA, 0, length(nombres)))
  colnames(Proy) <- nombres

  for (xmla in buscar) {
    foo <- Proy[1, ]

    tryCatch({
      xmlp <- DamePadre(xmla, "ContractFolderStatus")
```

```r
        foo$ContractID <- safe_xml_value(xmlp, "ContractFolderID")

        entry_node <- xmlParent(xmlp)
        id_value <- safe_xml_value(entry_node, "id")
        updated_raw <- safe_xml_value(entry_node, "updated")

        if (!is.na(id_value)) {
          bar <- strsplit(id_value, "/")[[1]]
          foo$entryID <- bar[length(bar)]
        }

        foo$updated <- tryCatch(as.Date(updated_raw), error = function(e) NA)

        # PlannedPeriod
        planned_period <- xmla[["PlannedPeriod"]]
        if (!is.null(planned_period)) {
          foo$start_date <- safe_xml_value(planned_period, "StartDate")
          foo$end_date <- safe_xml_value(planned_period, "EndDate")
          foo$duration <- safe_xml_value(planned_period, "DurationMeasure")
          foo$duration_unit <- if (!is.null(planned_period[["DurationMeasure"]])) {
            xmlGetAttr(planned_period[["DurationMeasure"]], "unitCode")
          } else {
            NA
          }
          foo$description <- safe_xml_value(planned_period, "Description")
        }

        # ContractExtension
        contract_ext <- xmla[["ContractExtension"]]
        if (!is.null(contract_ext)) {
          option_period <- contract_ext[["OptionValidityPeriod"]]
          foo$extension_description <- safe_xml_value(option_period, "Description")
        }

        Proy <- rbind(Proy, foo)
      }, error = function(e) {
        cat(sprintf("⚠  Error en archivo %s: %s\n", arch[ci], e$message))
      })
    }

    Proy
  }

  # Finalizar clúster
  stopCluster(cl)

  ### Limpieza y tratamiento de resultados en el DF Proy ------

  ## Eliminamos casos vacios
  Proy <- Proy %>%
    filter(!(is.na(start_date) & is.na(duration)))

  ## Damos formato a las variables

  Proy <- Proy |>
      mutate(
        start_date = as.Date(start_date),
        end_date = as.Date(end_date),
        updated = as.Date(updated),
        duration = suppressWarnings(as.numeric(duration)),
        description = na_if(trimws(description), ""),
        extension_description = na_if(trimws(extension_description), "")
      )
  ## Calculamos variable "duration" donde no la hay en formato MES (MON)
  Proy <- Proy %>%
    mutate(
      duration = ifelse(
        is.na(duration) & !is.na(start_date) & !is.na(end_date),
        floor(as.numeric(end_date - start_date) / 30),
        duration
      ),
      duration = ifelse(duration == 0, 1, duration),
      duration = as.integer(duration),
```

```r
      duration_unit = ifelse(
        is.na(duration_unit) & !is.na(start_date) & !is.na(end_date),
        "MON",
        duration_unit
      )
    )

  ## Pasamos todas las duraciones a meses (MON) - 30 dias y 12 meses
  Proy <- Proy %>%
    mutate(
      duration_m = case_when(
        duration_unit == "DAY" ~ floor(duration / 30),
        duration_unit == "ANN" ~ duration * 12,
        duration_unit == "MON" ~ duration,
        TRUE ~ NA_integer_
      ),
      duration_m = ifelse(duration_m == 0 & !is.na(duration), 1L, duration_m),
      duration_m = as.integer(duration_m)
    )

  ## Selecciono variables interesantes
  library(stringr)

  Proy <- Proy %>%
    transmute(
      Identificador = entryID,
      Identificador_lici = str_c(entryID, coalesce(as.character(ContractID), "NA"), sep = "//"),
      Fecha_actualizacion = updated,
      Plazo_m = duration_m,
      Prorroga = extension_description
    )

  ## Hacemos un unique
  Proy <- unique(Proy)

  ## En los casos que Identificador_lici sea el mismo, nos quedamos con el caso de Fecha_actualizacion mas tardia
  Proy <- Proy %>%
    group_by(Identificador_lici) %>%
    slice_max(order_by = Fecha_actualizacion, n = 1, with_ties = FALSE) %>%
    ungroup()


  # Guardar resultados
  save(Proy, nombres, file = RData_FILE).

  library(ggplot2)
  library(dplyr)
  library(openxlsx)
  library(RODBC)
  library(quanteda) #paquete recomendado en todos laos
  library(quanteda.textmodels) #aux
  library(quanteda.textstats) # aux
  library(readtext) # sencilla manera de leer data de texto en R
  library(spacyr) # NLP usando la libreria spaCy, incluyendo etiquetado part-of-speech, entity recognition y dependency parsing.
  library(devtools)
  library(zoo)
  library(stringr)
  # Ruta de trabajo
  setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP")
  #### Generamos data frame de Plazo de proyecto DF_plazo -----
  # Carga de archivos por año

  años <- 2012:2025
  lista_df <- list()

  for (año in años) {
    archivo <- paste0("3.34_PlannedPeriod_Extension_", año, ".RData")
    load(archivo)
    lista_df[[as.character(año)]] <- Proy
```

```r
}

# Unificación de datos en DF_plazo_u
DF_plazo <- bind_rows(lista_df)
DF_plazo_u <- unique(DF_plazo)  # Eliminamos duplicados
DF_plazo_u <- DF_plazo_u %>%
  group_by(Identificador_lici) %>%
  slice_max(order_by = Fecha_actualizacion, n = 1, with_ties = FALSE) %>%
  ungroup()

## Guardo resultados de plazo

save(DF_plazo_u, file="DF_plazo_u.RData")
write.xlsx(DF_plazo_u,"DF_plazo_u.xlsx", colNames = T, rowNames=F)
```