**Colour coding do not mean linkages between sections. Red means important. See Resources for itemised explanations.**

**Bold text are descriptions. The rest are codes. Underlines are segmented descriptions.**

# This is the user-interface definition of a Shiny web application.

# You can find out more about building applications with Shiny here:

#

# http://shiny.rstudio.com

#

**Our contacts for future team**

#By MrPotato for FYP AY2016/2017 Semester 3 Year 3. Contact Telegram @iamttl and @ShintoSamy

library(foreach) **For each loop, refer to resources. Attempted, did not use**

library(iterators) **Not loop. refer to resources. Attempted, did not use.**

library(shiny)

library(leaflet)

library(rgdal)

library(RMySQL)

#Function GetQuery with SQL and parameter inputs

getSQL <- function(SQL, **parameter = FALSE**){ **Default value of this parameter**

#Creation of database connection

dbhandle <- dbConnect(MySQL(),dbname="test",username="root", password="VFR4cde3")

print(parameter)

print(paste(SQL,collapse = ""))

if(any(parameter != FALSE)){

SQL <- **paste(SQL,collapse = "")**

}

result <- dbGetQuery(dbhandle, SQL)

cons <- dbListConnections(MySQL())

for(con in cons)

dbDisconnect(con)

print(result)

return(result)

}

**Call this function to get / execute sql.**

**How to use:**

1. **If statement do not require concatenation ie is fixed (SELECT \* FROM TABLE;) call by getSQL(statement).**
2. **Otherwise, with statement that can be varied (SELECT \* FROM TABLE WHERE TIME = TIME1) vs (SELECT \* FROM TABLE WHERE TIME = TIME2)**
   1. **Place the statement into a vector ie in this case, c(‘SELECT \* FROM TABLE WHERE TIME = ’,TIME)**
   2. **Note the variable TIME is another element in the c() vector. If statement continue, put it as another element, so on and so forth. Paste will concatenate the vector.**
   3. **Now the function call will be**
      1. **getSQL(c(‘SELECT \* FROM TABLE WHERE TIME = ’,TIME),TRUE)**
      2. **Note the TRUE parameter.**

**By right only one dbDisconnect will do, but actual run that is not enough. So to ensure ALL connections are disconnected a for loop is done on dbListConnections.**

#Creation of URA zone overlay

subzone <- readOGR(dsn = ".", layer = "URA subzone map final", verbose = F)

**Reading of the URA town planning zone files. Do not delete or modify unless clear purpose to.**

#Declaration of **Unix Time to Local Time conversion function**

convertUnixToLocalTime <- function(unixtime){

return(as.POSIXct(unixtime, origin="1970-01-01"))

}

**Planned to translate unix to readable time in R layer, however shifted to do it in SQL function.**

#Declaration of **Local Time to Unix Time conversion function**

convertLocalTimetoUnix<- function(localtime,**format = "%Y-%m-%d %H:%M:%S"**){ **Default**

return(as.POSIXct(strptime(localtime, format)))

}

#Setting up data binding from database for time dropdown

timequery <- getSQL("SELECT \* FROM TIMEDROPDOWNLIST;")

timequery2 <- getSQL("SELECT \* FROM TIMEDROPDOWNLIST;")

locationquery <- getSQL("SELECT LOCATION FROM COORDINATES;")

choicesloc <- setNames(locationquery$LOCATION,locationquery$LOCATION)

choices <- setNames(timequery$UNIX\_TIME,timequery$READ\_TIME)

choices2 <- setNames(timequery$UNIX\_TIME,timequery$READ\_TIME)

**Note how getSQL is called with fixed statement. This part alternatively can hard code drop down list choices (items) within R script. However maintenance wise will make this script overly cluttered. Database serves as single point of source.**

**Previous implementation to select available timing for starting and ending time from movement table - more accurate this way. But as movement table is expanded such selection will take a long time. Thus decided to have another table dedicated.**

**Same for location. Current implementation may cause start location not found in selected start time - No movement data originating from start location from select start time. (START LOCATION TEKONG, TIME 0300hrs)**

#Creation of Shiny UI Web Interface

ui <- bootstrapPage(

tags$style(type = "text/css", "html, body {width:100%;height:100%}"),

leafletOutput("map", width = "100%", height = "100%"),

absolutePanel(top = 10, right = 10,

selectInput("T1",

label = "Start Time (Where everyone at this time?):",

choices,selected = '1440000000'),

#uiOutput("areaSelector"), **<- Dynamic UI element. Not implemented. See below for respective renderUI pair.**

selectInput("LOC",

label = "At this location?:",

choicesloc,selected = 'Admiralty'),

selectInput("T2",

label = "Will be at this time?:",

choices2,selected = '1440003600')

)

)

**Definition of UI, drop down list etc. Here the timequery and location queries are used.**

**Tried to implement slider. However preliminary observation is that slider only allows for integer values == displayed text (v is d.t.). Thus not user friendly to display unix, or no obvious way to tweak and show string date time.**

**Possible to implement with further detailed explorations. Passed this idea in favor of main output.**

markerList <- iconList(

before = makeIcon(

iconUrl = "images/starhubgreen.png",

iconWidth = 45, iconHeight = 59

),

after = makeIcon(

iconUrl = "images/starhubred.png",

iconWidth = 45, iconHeight = 59

)

)

**Setting up of customised markers.**

#Creation of Server

server <- function(input, output, session) {

#output$areaSelector <- renderUI({}) **Dynamic UI. Not implemented. Stores element into output.**

#Calling server to output the map

output$map <- renderLeaflet({

**Point A** #Storing all the data in variable 'data' so that when the selectInput selects a timing,...

#...it will switch to the file that shows the timing

if(input$LOC == '.No Location' && input$T2 == '0'){

tabledata1 <- getSQL(c("SELECT START\_LOCATION, LONGITUDE AS STARTLONG, LATITUDE AS STARTLAT FROM MOVEMENT2, COORDINATES WHERE START\_LOCATION = LOCATION AND UNIX\_START = ",input$T1," GROUP BY START\_LOCATION"),TRUE)

}

**Feature removed. To show where everyone is at a given time. Single time point only.**

**Note how getSQL is called with variable parameters. Observe the differences.**

else if(input$LOC == '.No Location'){

tabledata1<-getSQL(c("select t1.movement\_id, t1.start\_location as START\_LOCATION, t1.startlong as STARTLONG, t1.startlat as STARTLAT, t2.dest\_location as DEST\_LOCATION, t2.destlong as DESTLONG, t2.destlat as DESTLAT, t1.count as count from

(select m.\*, coordinates.longitude as startlong, coordinates.latitude as startlat from movement2 m,coordinates where

unix\_start = ", input$T1, " AND

UNIX\_END = ", input$T2," AND

m.START\_LOCATION = coordinates.LOCATION and

longitude > 0) t1

**Getting start location lat lon coordinates**

inner join <- **joining two sql queries**

(select m.\*, coordinates.longitude as destlong, coordinates.latitude as destlat from movement2 m,coordinates where

unix\_start = ", input$T1, " AND

UNIX\_END = ", input$T2," AND

m.Dest\_location = coordinates.LOCATION AND

longitude > 0) t2

**Getting Dest location lat lon coordinates**

where

t1.movement\_id = t2.movement\_id;") **Join condition**

,TRUE)

**To show ALL movements from the selected start time from ALL locations. Will create a mess. In use.**

}else{

tabledata1<-getSQL(c("select t1.movement\_id, t1.start\_location as START\_LOCATION, t1.startlong as STARTLONG, t1.startlat as STARTLAT, t2.dest\_location as DEST\_LOCATION, t2.destlong as DESTLONG, t2.destlat as DESTLAT, t1.count as count from

(select m.\*, coordinates.longitude as startlong, coordinates.latitude as startlat from movement2 m,coordinates where

unix\_start = ", input$T1, " AND

UNIX\_END = ", input$T2," AND

START\_LOCATION = '", input$LOC, "' AND

m.START\_LOCATION = coordinates.LOCATION and

longitude > 0) t1

inner join

(select m.\*, coordinates.longitude as destlong, coordinates.latitude as destlat from movement2 m,coordinates where

unix\_start = ", input$T1, " AND

UNIX\_END = ", input$T2," AND

START\_LOCATION = '", input$LOC, "' AND

m.Dest\_location = coordinates.LOCATION AND

longitude > 0) t2

where

t1.movement\_id = t2.movement\_id;")

,TRUE)

}

**Selecting movements from a selected start time, to a selected end time, starting from a selected location.**

ipeople <- iter(tabledata1, by = "row") **Left over from attempt to do iterations.**

**Above code segment from point A till here are queries that will run with each user interaction. They prepares the data for display below.**

leaflet(subzone)%>%

addTiles() %>%

addPolygons(color = "white")%>%

addMarkers(data = tabledata1,~ STARTLAT,~ STARTLONG,popup = ~START\_LOCATION, clusterOptions = markerClusterOptions(), icon = ~markerList["before"])%>%

**Sequence of parameters.**

**(Data = dataset to use. Latitude, Longitude, popup, clustering options, marker icons)**

**‘~’ means the property (column) of the dataset passed in.**

addMarkers(data = tabledata1,~ DESTLAT,~ DESTLONG,popup = paste(sep="</br>",tabledata1$START\_LOCATION,tabledata1$DEST\_LOCATION,tabledata1$count), clusterOptions = markerClusterOptions(), icon = ~markerList["after"])%>%

**tabledata1$\*\*\*\* means the column of the data frame (dataset).**

**Refer to resource on addPolyline in leaflet.**

{

for(i in 1:nrow(tabledata1)){

. <- addPolylines(.,data = tabledata1[i,], c(tabledata1[i,]$STARTLAT,tabledata1[i,]$DESTLAT), c(tabledata1[i,]$STARTLONG,tabledata1[i,]$DESTLONG),

color = "green",popup = ~count, weight = ~count\*3)

}

return(.)

for(i in 1:nrow(tabledata1)){

content = paste(sep="</br>",tabledata1[i,]$START\_LOCATION,tabledata1[i,]$DEST\_LOCATION,tabledata1[i,]$count)

. <- addPopups(.,tabledata1[i,]$DESTLAT, tabledata1[i,]$DESTLONG, content,

options = popupOptions(closeButton = TRUE))

}

return(.)

}%>%

**Refer to resource on for loop in leaflet.**

**Attempted to add popup bubble for each movement, detailing the movement ie Count. However did not display. Not removed.**

#addPolylines(data = tabledata1[1:nrow(tabledata1),], c(tabledata1$STARTLAT,tabledata1$DESTLAT), c(tabledata1$STARTLONG,tabledata1$DESTLONG), color = "black", weight = 10)%>%

**Another way to create multiple poly line with each row of movements. Did not work out. Commented out.**

addProviderTiles("Thunderforest.Landscape", group = "Topographical") %>%

addProviderTiles("OpenStreetMap.Mapnik", group = "Road map") %>%

addProviderTiles("Esri.WorldImagery", group = "Satellite") %>%

addLegend(position = 'bottomright',opacity = 0.4,

colors = c('green','red','grey'),

labels = c('Starting Time','Ending Time','Screen Grey: No Destination Coordinate.'),

title = 'Spatial network analytics')%>%

addLayersControl(position = 'bottomright',

baseGroups = c("Topographical", "Road map", "Satellite"),

options = layersControlOptions(collapsed = FALSE))

})

**Ordering of layers will affect overlapping sequencing in output display. Those layers which comes later, ie addPolygon vs AddMarkers vs addPolyLines.**

**Polylines will overlap markers, markers will overlap polygons.**

**Within leaflet it is important to use %>% as line delimiter.**

}

shinyApp(ui, server) **Running the app**