**Pseudo Code:**

Class **Route\_Ranker\_Main**

{

Routes R = new Routes //Creates Routes object. A routes object will contain all possible routes presented to user.

R.findfirstRoute(source coordinates, destination coordinates) //Gets the best route suggested by Directions API, for the source and destination.

R.InsertnewJSON()

R.getRoutesHelper()

Points P=new Points

P.AssignRoutepoints(R)

P.SortRoutes()

Resultroutes=P.JSON.Routes //Send the array of Keys from hashmap P to results.

Return Resultroutes

}

Class **Routes**

{

JSON <Directions API Returned JSON, List (Derived Routes JSON)>

AllmodeTypes

**findfirstRoute**(source coordinates, destination coordinates) //Calls directions APi with source and dest and returns the JSON

{

Directions D=new Directions

JSON1=D.getDirections(Source cord, dest cord, Travel mode=transit,TransitModes=List(PublicModes))

//List(Modes) is list of public modes implemented

}

**InsertnewJSON**(JSON)

{

Modestypes=getDifferentModes(JSON) //Returns Different Mode combination string

JSONnew=JSON+Modestypes //

If JSONnew not in Hashset1

Insert JSONnew to Hashset1

}

**getDifferentModes**(JSON[])

{ Parse JSON[] to get the different modes of travel modes used.

Return ModeTypes.

}

**getRoutesHelper**(JSON)

{

For all JSONs in Hashset1

{

If Modestypes >=3

{

UpdateRoutes(JSON,JSON)

Threemode(JSON)

}

else

{

Twomode(JSON)

}

}

If Allmodetypes >= 2 Transit modes

{

Routenew=Overlay(JSON[])

UpdateRoutes(JSON,Routenew)

}

}

**UpdateRoutes**(JSON,Routenew)

{

Update the new derived Route to JSON route returned by getDirections into the Hashset.

}

}

Class **Modes**()

{

Num\_of\_modes

**Threemode**(JSON)

{

For all transit modes K in JSON

{

JSON2=Eliminate(K,JSON)

InsertnewJSON(JSON2)

}

}

**Twomode**(JSON)

{

Bike B=new Bike()

B.Addflag=B.checkcycleparameters(JSON)

Routenew1= B.CreateJSON(Addflag)

B.UpdateRoute(JSON, Routenew1)

For all transit modes K in JSON

{

Check if route exist for modes combinations of K replaced with other transit modes:

If true {

JSON3=Substitute(K,others)

InsertnewJSON(JSON3)

}

}

**Substitute**(JSON-modetypes,substitute\_with\_mode)

{

Query=Change JSON-modetypes to substitute\_with\_mode

JSON4= getDirections(query, source, destination)

Return JSON4

}

**Eliminate**(JSON-modetypes, modetoeliminate)

{

Query=Eliminate modetoeliminate from JSON-modetypes

JSON5=getDirections(query,source,destination)

Return JSON5

}

**Overlay(JSON[])**

{

Statichandler S=new Statichandler

If Allmodetypes contain Bus

{

Busstops[] = getbuscords(JSON[])

Else if Allmodetypes contain Luas

{

Luasstops[] =getluasstops(JSON[]

}

For all elements in Busstop[]

// Get cords of Busstop and Luasstops such that transfer distance is minimum.

{

MinTransferdist[]=min(getDistancefunction(Busstop[i],Luasstop[j])

JSON5=getDirections(Busstop cords, Luas stop cords, Travel Mode**=**walk)

JSON6=CreateJSON(JSON5,JSON[])

UpdateRoutes(JSON,JSON6)

}

}

Class **Bikes**

{

SourceDocDist[]

DestinationDocDist[]

Boolean Atsource,Atdestination

Checkcycleparameters(JSON)

{

//To Get Additional distance if cycle inserted at source

Read coordinates of source and Transit start point.

Source\_doccords[],Sourcewalkdist=Min(getDistance(getDocscord(),sourcecord))

DestinationDoccords[],Destwalkdist=Min(getdistance(getDocs,cord(),transitstartcords))

CyclingSdist=getdistance(Source\_doccords[],DestinationDoccords[])

//To Get additional distance if cycle inserted at destination

Read coordinates of final destination and Transit end point.

Destto\_doccords[],Dest\_todoc\_walkdist=Min(getDistance(getDocscord(),destcord))

DestfromDoccords[],Dest\_fromdoc\_walkdist=Min(getdistance(getDocs,cord(),transitstartcords))

CyclingDdist=getdistance(Destto\_doccords[],DestfromDoccords[])

Calculation to decide position of cycling mode:

If (CyclingSdist+Destwalkdist+Sourcewalkdist) <= (CyclingDdist+Dest\_todoc\_walkdist+Dest\_fromdoc\_walkdist)

Set AtSource to True

InsertBike(Source,Source\_doccords[],DestinationDoccords[])

Else

Set AtDestination to True

InsertBike(Destination,Destto\_doccords[],DestfromDoccords[])

}

InsertBike(Walksourcecords, docsoursecords, docdestcords)

{

WalktoDocJSON=getDirections(Walksourcecords,Docsourcecords,Travelmode=walk)

CyclingJSON=getDirections(docssourcecords,docdestcords,Travelmode=Cycle)

WalkfromDocJSON=getDirections(docdestcords,Destinationcords,Travelmode=Walk)

JSON7=CreateJSON (JSON, WalktoDocJSON, CyclingJSON, WalkfromDocJSON)

UpdateRoutes(JSON,JSON7)

}

Class **Points**

{

HashMap2<Route string, List(Pollution AQI, Weather quality, Traffic delay, Final Points)>

AssignRoutepoints()

{

User U=new User

U.getUserPreferences()

If U.Pollutionflag is TRUE

Pollution AQI =getPollutionAPI(Route cords)

If U.weatherflag is TRUE

Weather quality =getWeatherAPI(Route cords)

If U.Trafficflag is TRUE

Traffic delay =getTrafficAPI(Route cords)

UpdateHashMap2()

//Points Calculation

For each Route in R parse the JSON to get the below:

TotalWalkDist

TotalCycleDist

NumberofBusTransfers

NumberofLuasTransfers

TotWalkpoints = Int(TotalWalkDist) \* Walkpoints

TotCyclepoints=Int(TotalcycleDist) \* Cyclepoints

TotBuspoints=NumberofBusTransfer \* Buspoints

TotLuaspoints=NumberofLuasTransfer \* Luaspoints

Routepoints= TotWalkpoints + TotCyclepoints + TotBuspoints + TotLuaspoints

UpdateHashMap2()

}

}

Class **User**

{

Userid

Boolean Pollution, Weather, Traffic

getUserPreference()

{

Read User preferences from Database.

Set the Boolean parameters accordingly

}

getPollutionAPI ()

{

Createquery using coordinates.

Post query and get AQI values for 5 coordinates in route.

If error in response fetch AQI from static database.

}

getWeatherAPI()

{

Createquery using coordinates.

Post query and get weather codes for 5 coordinates in route.

If error in response fetch static database.

}

getTrafficAPI()

{

Createquery using route.

Post query and get weather codes for 5 coordinates in route.

If error in response return default values.

}

Class **Directions**

{

Source cords[]

Dest cords[]

getDirections(Source cords[], Dest cords[],other query parameters)

{

Create query

JSON=GoogleDirectionsAPI(query)

Return JSON

}

getDistance(source cords[], Destinations cords[])

{

Logic to calculate distance

Return distance

}

Class Static handler

{

Getluasstops(JSON)

{

Read JSON to get Luas source and destination names/coordinates

Read static data to get coordinates of all luas stops between source and destination luas stops

Return list of stops in route

}

GetBusstops(JSON)

{

Read JSON to get the Route id, source and destination stop coordinates

Read static data to get all stop coordinates of that route id and identify source and destination stop ids.

Return the list of stop coordinates between the identified source and destination for that route.

}