yoGERT GIS Toolbox

Capstone 4G06

Module Interface Specification for yoGERT

Team 19,
Smita Singh, Abeer Alyasiri, Niyatha Rangarajan,
Moksha Srinivasan, Nicholas Lobo, Longwei Ye

February 8, 2023

1 Revision History

Date	Version	Notes
January 18, 2023	1.0	**Smita**:Generating Activity Locations Module, Main Module, Data Transformation Module. **Abeer**:Network Graph, Shortest Route, Alternative Route, Route Generation, Mapping. **Moksha**: Preprocessing, Data Transformation. **Longwei** Module Decomposition, **Niyatha** Generating episodes
February 5th, 2023	1.1	**Smita**: Updating Modules **Moksha**: Updating Preprocessing Modules, addressing issues 50 and 53 from Team 14's Review

2 Symbols, Abbreviations and Acronyms

See SRS

Contents

3 Introduction

The following document details the Module Interface Specifications for the yoGERT toolbox

Complementary documents include the System Requirement Specification and Module Guide. The full documentation and implementation can be found at `https://github.com/NicLobo/Capstone-yoGERT`.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by .

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding	
Behaviour-Hiding	Generate Episode Route Generation Fetch Activity Locations Mapping Main
Software Decision	Preprocessing Inputs Network Graph Shortest Route Alternative Route Data Transformation

Table 1: Module Hierarchy

6 MIS of Module Preprocessing

6.1 Module

Preprocessing Inputs

6.2 Uses

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
validateCSV	CSV	$\mathbb{B}$, CSV	InvalidInput IOError

6.4 Semantics

6.4.1 State Variables

None.

6.4.2 Environment Variables

IOmanager : it is the access point between the local memory and the application. It is used when saving files locally.

6.4.3 Assumptions

- Assume CSV file paths are valid when working within the same directory.

6.4.4 Access Routine Semantics

validateCSV(csvfile):

- transition: None
- output: $(\forall columns : str | file \in validateCSV(csvfile) : validateCols(columns)) \Rightarrow true$
- output: $(\forall rows : tuple(\mathrm{R}, \mathrm{R}, \mathrm{str}) | file \in normalizeCSV(csvfile) : validateData(rows))) \Rightarrow$ CSV)
- exception: $(\neg(((\forall columns : str | file \in validateCSV(csvfile) : validateCols(columns)) \Rightarrow$ InvalidInput

- exception: $(\neg(\forall rows : tuple(\mathrm{R}, \mathrm{R}, \mathrm{str}) | file \in normalizeCSV(csvfile) : validateData(rows))) \Rightarrow$ IOError)

6.4.5 Local Functions

normalizeCSV : removes any invalid gps and time points from the original CSV data validateCols : Ensures that data has latitude, longitude, and time columns

7 MIS of Module Network Graph

7.1 Template Module

Network Graph

7.2 Uses

None.

7.3 Syntax

7.3.1 Exported Constants

DISTANCETOLERANCE = 200 - tolerance distance (m) added to the radius of the circle for the mapped area that encapsulates all the input GPS coordinates.
EARTHRADIUS = 6371000 - earth's radius (m).

7.3.2 Exported Type

NetworkGraph = ?

7.3.3 Exported Access Programs

Name	In	Out	Exceptions
new NetworkGraph	tuple of (latitude: $\mathbb{R}$, longitude: $\mathbb{R}$), tuple of (latitude: $\mathbb{R}$, longitude: $\mathbb{R}$), seq of tuple of (latitude: $\mathbb{R}$, longitude: $\mathbb{R}$), seq of {drive, walk, bike}	NetworkGraph	InvalidMode
getNearestNode	tuple of (latitude: $\mathbb{R}$, longitude: $\mathbb{R}$)	$\mathbb{N}$	OutOfBoundsCoord

7.4 Semantics

7.4.1 State Variables

dist : $\mathbb{R}$ - distance in m for the radius of the network graph using GPS coordinates.
stcoord : tuple of (latitude: $\mathbb{R}$, longitude: $\mathbb{R}$) - starting GPS coordinates inputted by the user.
nodes : seq of $\mathbb{N}$ - network graph node.

edges : seq of tuple of (ℕ, ℕ, ℝ) - network graph edge defined by 2 nodes and weight extracted from.

7.4.2 Environment Variables

onlinenetworkdatabase : connection to online database to retrieve information layers of intersections, roads, and paths initialised within module to build the network graph.

7.4.3 Assumptions

- Assume the inputted GPS coordinates are valid as they are the output of another module.
- Assume the online network database is always accessible within 10 minutes.

7.4.4 Access Routine Semantics

new NetworkGraph(*startcoord, endcoord, stopcoords, networkmode*):

- transition: The *dist* is set by $finddistance(startcoord, endcoord, stopcoords)$ which is the largest distance between *startcoord* and any other coordinate given in the input. *stcoord* is set to *startcoord* and sets the rest of the state variables using extracted data from *onlinenetworkdatabase* upon creation of the object *NetworkGraph*.
- output: Creates a *NetworkGraph* object with the parameters *startcoord, endcoord, stopcoords, networkmode* and extracted data from *onlinenetworkdatabase*
- exception: $(\neg(networkmode \in \{drive, walk, bike\}) \Rightarrow$ InvalidMode))

getNearestNode(*coord*):

- transition: None
- output: nodes[i] where $i \in [0..|stopcoords| - 1]$
- exception: $(\neg(findhdistance(coord, stcoord) \leq$ dist$) \Rightarrow$ OutOfBoundsCoord))

7.4.5 Local Functions

finddistance : tuple of (latitude: ℝ, longitude: ℝ) × tuple of (latitude: ℝ, longitude: ℝ) × seq of tuple of (latitude: ℝ, longitude: ℝ) → ℝ

- Description: Computes largest distance using *findhdistance* between starting coordinate and another coordinate, either a destination coordinate or a stop coordinate.

findhdistance : tuple of (latitude: ℝ, longitude: ℝ) × tuple of (latitude: ℝ, longitude: ℝ) → ℝ

- Description: finds the distance between two GPS coordinates using Haversine formula.

8 MIS of Generate Episodes

8.1 Template Module

Generate Episodes

8.2 Uses

None.

8.3 Syntax

8.3.1 Exported Constants

N/A

8.3.2 Exported Type

Activityepisode = ?

8.3.3 Exported Access Programs

Name	In	Out	Exceptions
new Activityepisode	A full path to a .csv file containing GPS points (Start: Lat, Long, Time, Stop: Lat, Long, Time)	A full path to a .csv file containing preprocessed GPS data	invalidCSV

8.4 Semantics

8.4.1 State Variables

points : columns of csv containing ($\mathbb{R}$, $\mathbb{R}$, *datetime*) - latitude, longitude and timestamp for a point.
finalpoints : columns of csv containing ($\mathbb{R}$, $\mathbb{R}$, *datetime*) - latitude, longitude and timestamp for a point after data has been run with generateEpisodes(csvPath).
mode : enum of ($\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$) - modes for episodes, STOP = 0, WALK = 1, DRIVE = 10
s

8.4.2 Environment Variables

N/A

8.4.3 Assumptions

- Assume the values for latitude, longitude and time in the given file are valid.

8.4.4 Access Routine Semantics

new Activityepisode(*csvPath*):

- transition: A full path to a .csv file containing GPS points (Start: Lat, Long, Time, Stop: Lat, Long, Time) to a full path to a .csv file containing preprocessed GPS data.
- output: finalpoints[i] where $i \in [0..|len(points)| - 1]$
- exception: $(\neg(csvPath) \Rightarrow$ InvalidFile))

createSegments(*csvPath, title*):

- transition: The given csv file is parsed and trimmed based using the average stepsize of the given data points.
- output: points[i] where $i \in [0..|len(points)|-1] \land i = i+(points[-1]-points[0])/len(points)$
- exception: $(\neg(csvPath) \Rightarrow$ InvalidFile))

createVelocities(*csvPath*):

- transition: The given csv file is used to create another csv file with the added column velocity
- output: points.append(velocity[i] where $(sqrt((i.lat - (i+1).lat)^2 + (i.long - (i+1).long)^2)/(i.time - (i+1).time)$
- exception: $(\neg(csvPath) \Rightarrow$ InvalidFile))

generateEpisodes(*csvPath*):

- transition: Generate activity episodes for each mode change in *points* in the csv file.
- output: endMode[i] where $((velocity[i] = mode.WALK.valueandvelocity[i]mode.DRIVE.value) \land endMode = mode.WALK) \lor (endVel = mode.DRIVE.value \land endMode = mode.DRIVE) \lor endMode = mode.STOP$
- exception: $(\neg(csvPath) \Rightarrow$ InvalidFile))

9 MIS of Data Transformation Module

9.1 Module

Data Transformation Module

9.2 Uses

None

9.3 Syntax

9.3.1 Exported Constants

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
GenInput	A full path to a .csv containing travel episodes with mode detected	seq of tuple of (latitude: $\mathbb{R}$, longitude: $\mathbb{R}$)	invalidCSV

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

None

9.4.3 Assumptions

We assume that the input files are generated from the GenerateEpisodes module.

9.4.4 Access Routine Semantics

GenInput(travelEpisodes):

- transition: N/A
- output: collection of tuple of (latitude: $\mathbb{R}$, longitude: $\mathbb{R}$) provided by genHelper
- exception: invalidCSV

9.4.5 Local Functions

genHelper : Path of CSV file(String)

- Description : Parses file path and reads CSV file content and converts content and returns seq of tuple of $(\mathbb{R}, \mathbb{R}) \rightarrow tupleof(\mathbb{R}, \mathbb{R})$

10 MIS of Module Shortest Route

10.1 Template Module

Shortest Route

10.2 Uses

Network Graph

10.3 Syntax

10.3.1 Exported Constants

None.

10.3.2 Exported Type

ShortestRoute = ?

10.3.3 Exported Access Programs

Name	In	Out	Exceptions
new ShortestRoute	NetworkGraph, tuple of (latitude: $\mathbb{R}$, longitude: $\mathbb{R}$), tuple of (latitude: $\mathbb{R}$, longitude: $\mathbb{R}$), seq of {time, distance}	ShortestRoute	InvalidWeight, OutOfBoundsCoord

10.4 Semantics

10.4.1 State Variables

startnode : $\mathbb{N}$ - graph node nearest the starting GPS coordinate.
endnode : $\mathbb{N}$ - graph node nearest the destination GPS coordinate.

10.4.2 Environment Variables

None.

10.4.3 Assumptions

- Assume the inputted GPS coordinates are valid as they are the output of another module.

10.4.4 Access Routine Semantics

new ShortestRoute(*graph, startcoord, endcoord, weighttype*):

- transition: Set state variables $startnode, endnode := findnode(graph, startcoord)$, $findnode(graph, endcoord)$
- output: Creates *ShortestRoute* using *DijkstraAlg(graph, weighttype, startnode, endnode)*
- exception: $(\neg(weighttype \in \{time, distance\}) \Rightarrow$ InvalidWeight$) \vee (\neg(startnode \in seqofgraph.nodes) \Rightarrow$ OutOfBoundsCoord$) \vee (\neg(endnode \in seqofgraph.nodes) \Rightarrow$ OutOfBoundsCoord)

10.4.5 Local Functions

findnode : *NetworkGraph* $\times$ tuple of $(\mathbb{R}, \mathbb{R}) \rightarrow \mathbb{N}$

- Description : *NetworkGraph.getNearestNode*(GPS coordinate) using the *NetworkGraph* access routine the function returns node number.

DijkstraAlg : *NetworkGraph* $\times$ String $\times \mathbb{N} \times \mathbb{N} \rightarrow seqof\mathbb{N}$

- Description : using Dijkstra's Shortest Path Algorithm given graph with weighted edges, source node, and destination node find shortest path as a seq of nodes.

11 MIS of Module Alternative Route

11.1 Template Module

Alternative Route

11.2 Uses

Network Graph

11.3 Syntax

11.3.1 Exported Constants

None.

11.3.2 Exported Type

AlternativeRoute = ?

11.3.3 Exported Access Programs

Name	In	Out	Exceptions
new AlternativeRoute	NetworkGraph, tuple of (latitude: $\mathbb{R}$, longitude: $\mathbb{R}$), tuple of (latitude: $\mathbb{R}$, longitude: $\mathbb{R}$)	AlternativeRoute	OutOfBoundsCoord

11.4 Semantics

11.4.1 State Variables

startnode : $\mathbb{N}$ - graph node nearest the starting GPS coordinate.
endnode : $\mathbb{N}$ - graph node nearest the destination GPS coordinate.

11.4.2 Environment Variables

onlinetransitdatabase : connection to online database to retrieve information layers of bus stops initialised within module to deactivate inaccessible edges on the network graph.

11.4.3 Assumptions

- Assume the inputted GPS coordinates are valid as they are the output of another module.

11.4.4 Access Routine Semantics

new AlternativeRoute(graph, startcoord, endcoord):

- transition: Set $startnode, endnode, graph := findnode(graph, startcoord)$, $findnode(graph, endcoor$ *deactivateedges(graph)* at the beginning.
- output: Creates *AlternativeRoute* using pathfinder(graph, startcoord, endcoord).
- exception: $(\neg(startnode \in seqofgraph.nodes) \Rightarrow$ OutOfBoundsCoord$)\vee(\neg(endnode \in seqofgraph.nodes) \Rightarrow$ OutOfBoundsCoord)

11.4.5 Local Functions

findnode : *NetworkGraph* $\times$ tuple of $(\mathbb{R}, \mathbb{R}) \rightarrow \mathbb{N}$

- Description : *NetworkGraph.getNearestNode*(GPS coordinate) using the *NetworkGraph* access routine the function returns node number.

pathfinder : *NetworkGraph* $\times$ String $\times \mathbb{N} \times \mathbb{N} \rightarrow seqof\mathbb{N}$
deactivateedges : *NetworkGraph* $\times$ *onlinetransitdatabase* $\rightarrow NetworkGraph$

- Description : deactivate edges inaccessible by bus according to the exported information.

12 MIS of Module Route

12.1 Module

Route Generation

12.2 Uses

Shortest Route, Alternative Route, Data Transformation

12.3 Syntax

12.3.1 Exported Constants

None.

12.3.2 Exported Type

None.

12.3.3 Exported Access Programs

Name	In	Out	Exceptions
GenerateGraph	seq of tuple of (latitude: $\mathbb{R}$, longitude: $\mathbb{R}$), String: $\in \{drive, walk, bike\}$	seq of $\mathbb{N}$	-
GenerateShortest-Routes	seq of tuple of (latitude: $\mathbb{R}$, longitude: $\mathbb{R}$), $\mathbb{B}$, String: $\in \{time, distance\}$	seq of $\mathbb{N}$	-
GenerateAlternative-Routes	seq of tuple of (latitude: $\mathbb{R}$, longitude: $\mathbb{R}$), $\mathbb{B}$	seq of $\mathbb{N}$	-

12.4 Semantics

12.4.1 State Variables

routes : seq of seq of $\mathbb{N}$ - list of route segments consisting of traces of nodes. The list represent a connected path for a full episode.
graph : *NetworkGraph* - Created for the given GPS input

12.4.2 Environment Variables

None

12.4.3 Assumptions

- Assume the inputted GPS coordinates are valid as they are the output of another module.

12.4.4 Access Routine Semantics

GenerateGraph(gpscoords, mode):

- transition: Set *graph* := *NetworkGraph.new NetworkGraph(getstart(gpscoords), getend(gpscoords), getstops(gpscoords), mode)*
- output: None
- exception: None

GenerateShortestRoute(gpscoords, hasstops, weighttype):

- transition: Update routes with (hasstops ⇒ (∀ connection : tuple — connection ∈ getconnections(gpscoords) : routes —— ShortestRoute.new ShortestRoute(graph, connection[0], connection[1], weighttype))) — (¬ hasstops ⇒ routes —— ShortestRoute.new ShortestRoute(graph, getstart(gpscoords), getstops(gpscoords), weighttype))
- output: None
- exception: None

GenerateAlternativeRoute(gpscoords, hasstops):

- transition: Update routes with (hasstops ⇒ (∀ connection : tuple — connection ∈ getconnections(gpscoords) : routes —— AlternativeRoute.new AlternativeRoute(graph, connection[0], connection[1]))) — (¬ hasstops ⇒ routes —— AlternativeRoute.new AlternativeRoute(graph, getstart(gpscoords), getstops(gpscoords)))
- output: None
- exception: None

12.4.5 Local Functions

getstart : seq of tuple of $(\mathbb{R}, \mathbb{R}) \rightarrow tupleof(\mathbb{R}, \mathbb{R})$

- Description : returns first coordinate.

getend : seq of tuple of $(\mathbb{R}, \mathbb{R}) \rightarrow$ tuple of $(\mathbb{R}, \mathbb{R})$

- Description : returns last coordinate

getstops : seq of tuple of $(\mathbb{R}, \mathbb{R}) \rightarrow$ seq of tuple of $(\mathbb{R}, \mathbb{R})$

- Description : returns list of stop coordinates.

getconnections : seq of tuple of $(\mathbb{R}, \mathbb{R}) \rightarrow$ seq of tuple of (tuple of $(\mathbb{R}, \mathbb{R})$, tuple of $(\mathbb{R}, \mathbb{R})$)

- Description : returns list connections of GPS pairs.

13 MIS of Activity Location

13.1 Template Module

Activity Location

13.2 Uses

None.

13.3 Syntax

13.3.1 Exported Constants

None

13.3.2 Exported Type

ActivityLocation = ?

13.3.3 Exported Access Programs

Name	In	Out	Exceptions
new ActivityLocation	name: String, lat: Latitude (ℝ), lon: Longitude (ℝ)	Activity Location	

13.4 Semantics

13.4.1 State Variables

name : String of Activity Location name
lat : latitude of Activity Location (ℝ)
lon : longitude of Activity Location (ℝ)

13.4.2 Environment Variables

None.

13.4.3 Assumptions

- Assume the values for latitude, longitude and name provided are valid

13.4.4 Access Routine Semantics

None

14 MIS of Stop Point

14.1 Template Module

Stop Point

14.2 Uses

None.

14.3 Syntax

14.3.1 Exported Constants

None

14.3.2 Exported Type

StopPoint = ?

14.3.3 Exported Access Programs

Name	In	Out	Exceptions
new StopPoint	lat: Latitude ($\mathbb{R}$), lon: Longitude ($\mathbb{R}$)	StopPoint	

14.4 Semantics

14.4.1 State Variables

lat : latitude of Stop Point ($\mathbb{R}$)
lon : longitude of Stop Point ($\mathbb{R}$)

14.4.2 Environment Variables

None.

14.4.3 Assumptions

- Assume the values for latitude, longitude provided are valid

14.4.4 Access Routine Semantics

None

15 MIS of Generating Activity Locations

15.1 Module

Fetch Activity Locations

15.2 Uses

~~Uses no other modules~~ Activity Location, Stop Point

15.3 Syntax

15.3.1 Exported Constants

None.

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
fetchStopAL	collection of Stop Points	collection of tuples (Stop Point, Activity Location)	EmptyListException

15.4 Semantics

15.4.1 State Variables

Not applicable

15.4.2 Environment Variables

onlinenetworkdatabase: connection to online database to retrieve information layers of intersections, roads, and paths, and activity locations.

15.4.3 Assumptions

- Collection of stop locations longitudes and latitudes that are correct and valid
- Tolerance of 25 meter radius is appropriate for finding activity locations
- Onlineworkdatabase is accurate and accessible within 10 minutes

15.4.4 Access Routine Semantics

fetchStopAL(listOfStops)():

- transition: Not applicable
- output: collection of tuples (Stop Point, Activity Location)
- exception: $listOfStops = \{\} \Rightarrow$ EmptyListException

15.4.5 Local Functions

fetchActivityLocations : tuple of (latitude: $\mathbb{R}$, longitude: $\mathbb{R}$)

- Description: Computes activity location given the latitude and longitude of a specific stop location by fetching activity locations in a 25 meter radius from onlinework-database and ~~returns list of activity location names and latitudes and longitudes~~ returns list of ActivityLocation objects

16 MIS of Module Mapping

16.1 Template Module

Mapping

16.2 Uses

Network Graph, Shortest Route, Alternative Route

16.3 Syntax

16.3.1 Exported Constants

None.

16.3.2 Exported Type

Display = ?

16.3.3 Exported Access Programs

Name	In	Out	Exceptions
new Mapping	*NetworkGraph*, seq of *ShortestRoute*, seq of *AlternativeRoute*, seq of $\mathbb{N}$, seq of $\mathbb{N}$	HTML FILE	-
updateMapping	seq of *ShortestRoute*, seq of *AlternativeRoute*, seq of $\mathbb{N}$, seq of $\mathbb{N}$	HTML FILE	-

16.4 Semantics

16.4.1 State Variables

map : HTML FILE - displays mapped routes and points.

16.4.2 Environment Variables

IOmanager : it is the access point between the local memory and the application. It is used when saving and updating HTML files locally.

16.4.3 Assumptions

- Assume all the inputs are valid as they are outputs of previous modules.

16.4.4 Access Routine Semantics

new Mapping(graph, shortest, alternative, activitylocations, episodes):

- transition: None.
- output: Create a *Mapping* object and saves it locally with *IOmanager*
- exception: None

updateMapping(shortest, alternative, activitylocations, episodes):

- transition: Add new elements to *Mapping* object and saves it locally with *IOmanager*
- output: None.
- exception: None

16.4.5 Local Functions

getNode : seq of tuples of $(\mathbb{R}, \mathbb{R}) \Rightarrow$ seq of $\mathbb{N}$

- Description: converts GPS coordinates to graph nodes.

17 MIS of Main Module

17.1 Module

Main

17.2 Uses

Input Processing Module, Episode Generation and Mode Detection Module, Activity Location Module, Route Generation Module, Data Transformation Module

17.3 Syntax

17.3.1 Exported Constants

N/A

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
inputProcessing	$\mathbb{N}$, seq of String	.SHP files	InvalidCoord
episodeGeneration-ModeDetection	CSV file	CSV file	InvalidFile
findActivityLocations	CSV file	CSV file	EmptyListException
generateGraph	CSV file, String $\mathbb{B}$	$\mathbb{B}$	InvalidMode, OutOfBoundsCoord
generateShortestPath	CSV file, String $\mathbb{B}$	seq of $\mathbb{N}$	InvalidWeight, OutOfBoundsCoord
generateAlternative-Path	CSV file $\mathbb{B}$	seq of $\mathbb{N}$	OutOfBoundsCoord
mapEpisodes	NetworkGraph, CSV file	HTML file	-
mapActivityLocations	NetworkGraph, CSV file	HTML file	-
mapSRoute	NetworkGraph, CSV file	HTML file	-
mapARoute	NetworkGraph, CSV file	HTML file	-

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

18 Appendix

Extra information if required