# Table of contents

[Table of contents 1](#_Toc437237257)

[List of Table 4](#_Toc437237258)

[List of Figures 8](#_Toc437237259)

[Definitions, Acronyms, and Abbreviations 10](#_Toc437237260)

[A. Introduction 1](#_Toc437237261)

[1. Project Information 1](#_Toc437237262)

[2. Introduction 1](#_Toc437237263)

[3. Current Situation 1](#_Toc437237264)

[4. Problem Definition 2](#_Toc437237265)

[5. Proposed Solution 2](#_Toc437237266)

[5.1. Feature functions 2](#_Toc437237267)

[5.2. Advantages and disadvantages 4](#_Toc437237268)

[6. Functional Requirements 5](#_Toc437237269)

[7. Role and Responsibility 5](#_Toc437237270)

[B. Software Project Management Plan 6](#_Toc437237271)

[1. Problem Definition 6](#_Toc437237272)

[1.1. Name of this Capstone Project 6](#_Toc437237273)

[1.2. Problem Abstract 6](#_Toc437237274)

[1.3. Project Overview 6](#_Toc437237275)

[2. Project organization 9](#_Toc437237276)

[2.1. Software Process Model 9](#_Toc437237277)

[2.2. Roles and responsibilities 10](#_Toc437237278)

[3. Project Management Plan 11](#_Toc437237279)

[3.1. Software development life cycle 11](#_Toc437237280)

[3.2. Phase Detail 14](#_Toc437237281)

[C. Software Requirement Specification 16](#_Toc437237282)

[1. User Requirement Specification 16](#_Toc437237283)

[1.1. Customer requirement 16](#_Toc437237284)

[1.2. Staff requirement 16](#_Toc437237285)

[2. System Requirement Specification 17](#_Toc437237286)

[2.1. External Interface Requirement 17](#_Toc437237287)

[2.2. System Overview Use Case 17](#_Toc437237288)

[3. Software System Attribute 19](#_Toc437237289)

[3.1. Usability 19](#_Toc437237290)

[3.2. Reliability 19](#_Toc437237291)

[3.3. Availability 19](#_Toc437237292)

[3.4. Security 19](#_Toc437237293)

[3.5. Maintainability 19](#_Toc437237294)

[3.6. Portability 19](#_Toc437237295)

[3.7. Performance 19](#_Toc437237296)

[4. Conceptual Diagram 19](#_Toc437237297)

[D. Software Design Description 22](#_Toc437237298)

[1. Design Overview 22](#_Toc437237299)

[2. System Architecture Design 22](#_Toc437237300)

[3. Component Diagram 23](#_Toc437237301)

[4. Detailed Description Class Diagram 23](#_Toc437237302)

[4.1. Web System class diagram 23](#_Toc437237303)

[4.2. Web class diagram 24](#_Toc437237304)

[5. Interface 27](#_Toc437237305)

[5.1. Component interface 27](#_Toc437237306)

[6. Database design 30](#_Toc437237307)

[6.1. Main Entity Relationship diagram 30](#_Toc437237308)

[6.2. Sub entity Relationship diagram 31](#_Toc437237309)

[7. Algorithm 31](#_Toc437237310)

[7.1. Raptor algorithm 31](#_Toc437237311)

[7.2. Search bus two point: 35](#_Toc437237312)

[7.3. Search bus more than two points 38](#_Toc437237313)

[7.4. Search bus more than two points with optimize 40](#_Toc437237314)

[7.5. Search motorbike more than two points 41](#_Toc437237315)

[7.6. Search motorbike more than two points optimize 44](#_Toc437237316)

[7.7. Distance from one point to segment algorithm 46](#_Toc437237317)

[7.8. Notify algorithm when on right route. 47](#_Toc437237318)

[7.9. Notify algorithm when on wrong route 49](#_Toc437237319)

[7.10. Download audio file algorithm 52](#_Toc437237320)

[2. Database Relationship Diagram 54](#_Toc437237321)

[A. Appendix 55](#_Toc437237322)

# List of Table

[Table 1: Definitions, Acronyms, and Abbreviations 7](#_Toc437238028)

[Table 2: Roles and Responsibilities 5](#_Toc437238029)

[Table 3: Hardware requirement for continuous integrating server 8](#_Toc437238030)

[Table 4: Hardware requirement for web development 8](#_Toc437238031)

[Table 5: Hardware requirement for mobile development 9](#_Toc437238032)

[Table 6: Hardware requirement for wear development 9](#_Toc437238033)

[Table 7: Software requirement 9](#_Toc437238034)

[Table 8: Roles and responsibilities 11](#_Toc437238035)

[Table 10:Software development life cycle 13](#_Toc437238036)

[Table 11: Requirements definition 14](#_Toc437238037)

[Table 12: System and software design 14](#_Toc437238038)

[Table 13: Implementation and unit test 14](#_Toc437238039)

[Table 14: Integration and system testing 15](#_Toc437238040)

[Table 15: Operation and maintenance 15](#_Toc437238041)

[Table 56 Class dictionary 24](#_Toc437238042)

# List of Figures

[Figure 1: Waterfall model 10](#_Toc437238043)

[Figure 2: System overview use cas 18](#_Toc437238044)

[Figure 43: Conceptual Diagram 20](#_Toc437238045)

[Figure 44: Mobile Conceptual diagram 21](#_Toc437238046)

[Figure 45 System architecture design 22](#_Toc437238047)

[Figure 46 Component Diagram 23](#_Toc437238048)

[Figure 47: Web system class diagram 23](#_Toc437238049)

[Figure 60: Activity diagram - <Guest> Route street 26](#_Toc437238050)

[Figure 81: Search bus two point algorithm – flowchart 37](#_Toc437238051)

[Figure 82: Search bus more than two points algorithm – flowchart 39](#_Toc437238052)

[Figure 83: Perpendicular example 47](#_Toc437238053)

# Definitions, Acronyms, and Abbreviations

|  |  |
| --- | --- |
| Name | Definition |
| SWR | Smart Wear on your Route |
| API | Application Programming Interface |
| Wear device | Smart watch that uses Android Wear OS 4.4 or above |
| Staff | Administrator of website |
| BusMap | Official mobile application developed by Ho Chi Minh Ministry of Communications and Transport -http://www.buyttphcm.com.vn/Detail\_News.aspx?sl=717 |
| RAPTOR | Round-based Public Transit Optimized Router |
| mcRAPTOR | More criteria RAPTOR |
| Station | Distinct location in the network where one can board or get off a vehicle (bus, train) |
| Round | Represents a sequence of stations a specific vehicle (train, bus, subway …) |
| Connection | A connection models a vehicle departing at one start station to end station of one trip without intermediate halt. |
| Trip | Trip is a round with time arrival information at each station. One trip often has many routes. |
| Footpath | Model walking connection between stations. |
| Pareto Set | A subset of the set of feasible points of solutions that contains all points that have at least one objective optimized while holding all other objectives constants. |

Table 1: Definitions, Acronyms, and Abbreviations

# 

# Introduction

## Project Information

Project name: Smart Wear on Your Route

Project Code: SWR

Product Type: Website, Android and Android Wear application

Start Date: September 7th, 2015

End Date: December 20th, 2015

## Introduction

Nowadays, within the strong development of presently economy, time is always one of the priorities in all areas. In particularly, when participating in traffic, how to know fastest route in the journey is the critical condition for user.

Presently, most of an application on market is not support searching through more than two points. For example, Google Map and BusMap just supports on searching through two points at most so that they cannot help user if user has more than one place to go. Moreover, no mobile application supports routing when participating traffic, for example notify message when customer near a station, or need to turn on specific street. Last but not least, no mobile application supports wear devices, so user must look up their mobile phone when participating in traffic and this behavior makes some inconveniences such as thief, accident …

Facing above problems, our team build the application named is “Smart Wear on Your Route”. In our application, we allow user find route through more than two points. We also support user choose their departure time so they can choose suitable route that they can come to place on time. We support user search by using voice command. We provide notify system function allowing customer know when participating traffic. We enhance this notify system by using sound notification when for user when participating traffic. We also provide wrong route detection system when user go to wrong route. We support map offline, so that customer doesn’t need network while participating traffic. Moreover, our application supports wear devices so user can look up on their wear device when they participating traffic avoiding some above problems. However, our system still works properly on mobile when wear has problems or doesn’t connected to mobile when participating traffic.

In additional, we also provide system software on website for staff to manage bus route, bus time information and approve the change from background handler.

## Current Situation

Nowadays, when participating in traffic, user often wants to find route through some locations. This situation becomes more important especially participating by bus or motorbike. Currently, mobile market has some applications that support routing such as Google map or BusMap.

Google Map and BusMap allow user enters starting location and ending location into their cell phones (with already networked). After that, Google Map and BusMap will suggest some optimal paths. Finally, user will choose the best route suitable for their need. When user finish selected their choice, mobile application will render route on mobile screen so user can follow the route.

With Google Map, user can optional enter arrival time and departure time. By this constraint, Google Map will find suitable routes that user can start and come to place on time.

## Problem Definition

Below are disadvantages of current situation:

* BusMap doesn't support motorbike route.
* BusMap doesn’t support time constraint (arrival time, departure time) when finding route.
* Google Map and BusMap don't support searching through more than two points.
* Google Map and BusMap don’t support routing when participating traffic.
* Google Map and BusMap don’t support sound routing when participating traffic.
* Google Map supports map offline with some limitations: Download area is limit with 50km x 50km and will be expired after 30 days. After 30 days, user must download that area again.
* Google Map voice search has some limitations such as just can only search one place.
* BusMap and Google Map don’t support using smart watch for routing when participating traffic, just for phone. There are some disadvantages of using only smartphone to find route such as theft, inconvenience, no safety in motorbike control as well as the bus.

## Proposed Solution

Our proposed solution is to build and mobile application and android wear application named “Smart Wear on Your Route” to resolve the current situations. We also design the system to be scalable so we can extend our system for more platforms (iOS, Windows Phone) in the future and can be used for more transit protocols (train, high-speed train)

SWR system includes a web application, background process, mobile application and wear application with following functions:

### Feature functions

Web application: design for staff. Those functions support staff manage route and bus timetable. So Staff can approve or reject data change before saving to database.

* Manage routing: staff edits information for bus route and bus timetable.
* Notify new update data from server to staff: if official website (http://www.buyttphcm.com.vn/) has new data, background process will notify to staff and staff will decide approve this update or not.

Background process: process for checking data for getting latest data and update to server.

* Check new data periodically: Background process will check new data at 0 AM each day. If background process detects that data has been changed, background process will write new data to temporary database and notify messages for staff.

Mobile application: developing functions on mobile supports user search bus route or motorbike route both using voice or typing. Mobile functions also support user route when participating traffic using voice or text. Also mobile functions will support user recommend right route when they are on wrong way.

* Searching bus route through from two points to four points: user inputs start point, two optional middle points and end point and optional departure time. Application will find the best bus route from start point through middle points to end point which optimize condition (shortest time, least number change route)
* Find bus route through from two points to four points with optimize: user inputs start point, two optional middle points and end point and optional departure time, then choose “optimize” option. Application will find the best bus route from start point through three points which optimize condition (shortest time, least number change route), no matter order last three points.
* Find motorcycle route through from two points to four points: user inputs start point, two optional middle points and end point and optional departure time. Application will find the best motorcycle route from start point through middle points to end point which optimize condition shortest time.
* Find motorcycle route through from two points to four points with optimize: user inputs start point, two optional middle points and end point and optional departure time, then choose “optimize” option. Application will find the best motorcycle route from start point through three points which optimize condition shortest time, no matter order last three points.
* Assist user searching using typing or voice command: User has two ways for input data: Using keyboard and typing text or using voice command.
* Routing bus route: when user chooses one bus route for participating traffic, application will start for tracking user’s location, and notify message (by vibrate, notification, sound) when user is near a bus station that need to get off station.
* Routing motorbike route: when user chooses one motorbike route for participating traffic, application will start for tracking user’s location, and notify message (by vibrate, notification, sound) when user is near a turn. Application also notify message when user goes to wrong route and recommmend suitable message when come to searched route again.
* Detect wrong motorbike route: When users go to wrong route, application will detect and notify message periodically for users know that they have gone wrong route.
* Recommend suitable motorbike direction for user when user is near searched route: when user is on wrong route and go near again searched route, application will recommend suitable direction for user.
* Map offline: By using map offline, user can use routing function without network connectivity.

Wear app: functions developing on wear supports user easily navigate searched route (i.e: when participating traffic, so user doesn’t need to open mobile).

* Bus: Application will notify for user when bus nears the station that user should to leave:  if bus in circular range of station of the route's plan, application will show the message name of the next station.
* Motorcycle:
* App will notify when user has to turn route: If user drives in circular range of next turn, application will automatically show message which should to do next and vibrate until user out of this range.
* App will notify when user goes to wrong route: If user goes to wrong route compare to original searched route, application will automatically show message that user has gone to wrong route.
* Map:
* Show your current location: show current user location on map with route user should to go (including bus or motorbike).

### Advantages and disadvantages

Advantages:

* Support optimization passes through multi points.
* Support bus route timetable so that user can easily find suitable round for their time.
* Support routing on street both on motorbike and bus.
* Support voice routing on street, both on motorbike and bus.
* Support two methods for searching place: using text typing or using voice.
* Support detect wrong route and notify to user when user uses motorbike.
* Support recommend new route when user is from wrong way and is near again searched route.
* Support map offline so user doesn’t need network connectivity when participating traffic.
* Support on smart wear. User can look up information easier than using mobile phone. Also using smart wear decreases ability for mobile phone to be stolen.

Disadvantages:

* + Smart wear has higher price than smart phone and these screen is slightly small and hard to use for newbie.
* Application must parse data from third website so server’s data cannot be updated intermediately.

## Functional Requirements

Web Component: (for staff only):

* Edit bus route and bus time information.
* Approve bus route and bus time change from background handler and write to official database.

Parser Component:

* Parse bus route information.
* Parse bus timetable information.
* Periodically, detect the change from official bus website in order to write to temporary database.

Mobile Component:

* Find the path’s optimization from two points to four points when using bus.
* Find the path’s optimization from two points to four points when using motorbike.
* Voice search for bus and motorbike.
* Routing when participating traffic both on bus and motorbike vehicle.
* Detecting wrong route and recommend again true direction.
* Sync data from mobile to wear.

Wear Component:

* Receive data from mobile.
* Notify message when user near the bus station that should to left.
* Notify next turns when user drive by motorbike.
* Auto scroll to current user’s location on map.

## Role and Responsibility

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Full Name | Role | Position | Contact |
| 1 | Kiều Trọng Khánh | Project Manager | Supervisor | khanhkt@fpt.edu.vn |
| 2 | Huỳnh Quang Thảo | Developer | Leader | huynhquangthao@gmail.com |
| 3 | Trần Thanh Ngoan | Developer | Member | ngoanttse61125@fpt.edu.vn |
| 4 | Nguyễn Trung Nam | Developer | Member | namntse61132@fpt.edu.vn |
| 5 | Ngô Tiến Đạt | Developer | Member | datntse60980@fpt.edu.vn |

Table 2: Roles and Responsibilities

# Software Project Management Plan

## Problem Definition

### Name of this Capstone Project

Official name: Smart Wear on Your Route

Vietnamese name: Hỗ trợ đi đường với thiết bị đeo tay thông minh.

Abbreviation: SWR

### Problem Abstract

The transport system has a lot of roads and bus routes. Even local people hardly know well. With this system, users must spend a lot of time to know how to get from one location to others. With the worst case scenario, user can lose their phone or cause danger to themselves because look up mobile phone frequently when participating traffic.

We provide application, which helps users find bus routes or motorcycle routes from two points to four points with optimized conditions (shortest time, walking distance, number transfers and departure time). Application supports finding route on android phone and real-time navigation on mobile and smart wear. Application also supports detect and recommend suitable direction when user is on wrong way. Application supports map offline so user doesn’t need network connectivity when participating traffic. Application also supports voice speech command.

### Project Overview

#### Current Situation

Below are the problems encountered in this project:

* **Depend on other system**: all the crawl data is gotten from other system. So if data has been changed, our system will be out of update.
* **Network Connection:** mobile application must be connected to 3G for getting GPS and map information periodically. Moreover, wear device must be in range of Bluetooth because pairing with mobile device.
* **Testing:** hardly to test GPS function when switching between bus stations due to physical limitation.
* **Absent of team members:** team members get sick or unexpected problems. Working time and learning time are different among members.
* **Violence Google terms of Service:** Google API Terms of Service doesn’t allow real-time navigation or route guidance, including but not limited to turn-by-turn route guidance that is synchronized to the position of a user's sensor-enabled device.

#### The Proposed System

Based on current issues, combined with research results in HCM city traffic system, we propose a system allow user searches route on mobile phone and real-time navigation by using smart wear.

We also developed algorithm name RAPTOR algorithm[[1]](#endnote-1) for finding shortest distance between two points on graph.

We also store a local database on mobile device to allow users search bus route information offline.

Moreover, we have built a background handler for checking third-party server periodically every 0AM to always get the latest data.

Our system includes three main subsystems: an online website for staffs, a mobile application as well as watch application for Participants traffic.

##### Website

Website provided following features:

* For staffs:
* Staff edits information for bus route and bus timetable.
* Notify new update data from server to staff.
* Beside above, website system also provides an API interface for mobile, wear applications to retrieve data.

##### Background Handler

Check new data periodically at 0 AM each day.

Parse data from website.

##### Bus Routing Mobile Application

This application is used by user and does followings:

* Find bus route from two points to four points.
* Find bus route from two points to four points with optimize.
* Find motorcycle route from two points to four points.
* Find motorcycle route from two points to four points with optimize.
* Routing for motorbike route.
* Routing for bus route.
* Detect wrong way and recommend true direction for user.
* Notification for user to know next bus route or motorbike turn.

##### Wear Application

Application for wear device must be paired with android phone and do following function:

* Bus: Application will notify for user when bus nears the station that user should to leave.
* Motorcycle: Application will notify when user has to turn route.
* Map: Show current user location on map with route user should to go (including bus or motorbike.

#### Boundaries of the System

Our system is working on Ho Chi Minh city transportation system.

The language for mobile application and wear application is Vietnamese. The language for staff management site is English.

Mobile devices run android 4.3 or above. Smart wears run android wear API 20 or above.

The complete product includes:

* Website application for staffs.
* Background handler.
* Mobile and wear application for users.

#### Future plans

Our current system only supports bus and motorbike route in Ho Chi Minh city. We also just optimize time travel, walking distance, number transfers and departure time. We design the system to make it easier for further development including:

* **Support more public transportation**: system will support more public transits such as train andhigh-speed train.
* **Support more conditions**: system will provide more options for user choice such as: minimum ticket price, ticket price (student, old people). Moreover, we will design system support user give both departure time and arrival time.
* **Support other cities in Viet Nam**: system will support user find routes in other big cities such as Ha Noi or Da Nang.

#### Development Environment

##### Hardware requirement

For continuous integrating server:

|  |  |  |
| --- | --- | --- |
| Hardware | Minimum Requirements | Recommended |
| Internet Connection | 512Kbps | 8 Mbps |
| Operating System | Ubuntu Server 12 LTS | Ubuntu Server 14.04.2 LTS |
| Computer Processor | Intel® Core 2 Duo | Intel® Core(TM) i5 CPU , M 460 @ 2.53GHz |
| Computer Memory | 1GB RAM | 3GB or more |

Table 3: Hardware requirement for continuous integrating server

For web development:

|  |  |  |
| --- | --- | --- |
| Hardware | Minimum Requirements | Recommended |
| Internet Connection | 512Kbps | 8 Mbps |
| Operating System | Window Vista, 7, 8 | Window 7, 8 |
| Computer Processor | Intel® Core 2 Duo | Intel® Core(TM) i5 CPU , M 460 @ 2.53GHz |
| Computer Memory | 4GB RAM | 6GB or more |

Table 4: Hardware requirement for web development

For mobile development:

|  |  |  |
| --- | --- | --- |
| Hardware | Minimum Requirements | Recommended |
| Internet Connection | Wi-Fi Connection 512Kbps  Bluetooth Connection 4.0 | Wi-Fi Connection 8Mbps  Bluetooth Connection 4.0 |
| Operating System | Android 4.3 | Android 5.0 |
| Hardware | GPS supported | GPS supported |
| Memory | 1 GB RAM | 2 GB or more |

Table 5: Hardware requirement for mobile development

For wear development:

|  |  |  |
| --- | --- | --- |
| Hardware | Minimum Requirements | Recommended |
| Internet Connection | Bluetooth Connection 4.0 | Bluetooth Connection 4.0 |
| Operating System | Android Wear API 20 | Android Wear API 22 |
| Hardware |  | GPS supported |
| Memory | 512MB RAM | 512 MB or more |

Table 6: Hardware requirement for wear development

##### Software requirement

|  |  |
| --- | --- |
| Software | Name / Version |
| Operating system | Windows 7 or above, MacOS 10.10 or above |
| Environment | JDK 1.6, Java EE 6, Android SDK minimum API 20 |
| Modeling tool | StartUML 5.0 |
| IDE | Intellij IDEA 14.1, Android Studio 1.3.1 |
| DBMS | MySQL 5.6 |
| Source control | Git 2.3.2, Source Tree 1.6.20.0 |
| Web browser | Chrome 42 or above |
| Team Collaboration | Slack 1.1.3 |
| Issues and Task Management | GitHub and ZenHub |
| Others | Microsoft Word, Microsoft Excel, Adobe Photoshop |

Table 7: Software requirement

## Project organization

### Software Process Model

This project is developed under waterfall model. We apply customized waterfall model to capable with current situation in our team. We choose this model because the following reasons:

* This project is 4 months long due to the FPT University Capstone Project timeline, which can be consider a short project.
* Based on discussing carefully before with our supervisor, the requirements of this project are stable, clear, fixed and well understood by all team members.
* This project use android and wear technology, which we have strong background knowledge and well practice skills. Moreover, google has enough tutorials, supporting for further research.



Figure 1: Waterfall model

Reference: Page 30, chapter 2, Software process model, SOFTWARE ENGINEERING 9th Edition, by Ian Sommerville.

We customize the waterfall model from the reference to make the process more capable with current situation of our team.

### Roles and responsibilities

|  |  |  |  |
| --- | --- | --- | --- |
| No | Full name | Role in Group | Responsibilities |
| 1 | Kiều Trọng Khánh | Supervisor / Project Manager | - Clarify user requirement.  - Technical support and business analysis.  - Tracking development process.  - Review document and product. |
| 2 | Huỳnh Quang Thảo | Team leader, BA, Developer, Tester | - Tracking process.  - Planning project, distribute tasks.  - Requirement analysis.  - Database design.  - Documentation.  - GUI Design.  - Coding.  - Testing.  - Deploy product. |
| 3 | Trần Thanh Ngoan | BA, Developer, Tester | - Requirement analysis.  - Database design.  - Documentation.  - GUI Design.  - Coding.  - Testing. |
| 4 | Nguyễn Trung Nam | BA, Developer, Tester | - Requirement analysis.  - Database design.  - Documentation.  - GUI Design.  - Coding.  - Testing. |
| 5 | Ngô Tiến Đạt | BA, Developer, Tester | - Requirement analysis.  - Database design.  - Documentation.  - GUI Design.  - Coding.  - Testing. |

Table 8: Roles and responsibilities

## Project Management Plan

### Software development life cycle

Below are all the major tasks that need to be performed sequentially during the development of the system.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Phase | Description | Deliverables | Resource needed | Dependencies and Constrains | Risk |
| Requirements Definition | Identify and clarify system requirements. | Report No.1 Introduction. | 20 man-days | N/A | - Missing requirement.  - Project’s scope can be unclear.  - Lack of member share and understand. |
| System and Software Design | - Identify hardware and software requirements.  - Decide software architect and clarify software detail design.  - Design database. | Report No.2 Software Project Management Plan, Report No. 3 Software Requirement Specification and  Report No. 4 Software Design Description. | 50 man-days | Depend on Requirements Definition. | - Misunderstood or unclear system’s requirement.  - Lack of practical experience leading to unreasonable design. |
| Implementation and Unit Testing | - Implements all functions of system.  - Create test plan.  - Perform Unit testing. | Software package. | 120 man-days | - Base on Software Requirement Specification and Software Design Description.  - Coding try to follow coding convention. | - Member does not performs unit test.  - Lack of practical experience. |
| Integration and System Testing | - Perform integration test and system test. | Report No. 5 System Implementation & Test | 35 man-days | Implementation and Unit Testing are finished. | - Lack of testing experience leading to lack of test cases.  - Not enough time for performing test. |
| Operation and Maintenance | - Deploy the system  - Create the user’s manuals.  - Do routine maintenance activities. | Report No.6 Software User’s Manual | 15 man-days | Integration and System Testing are finished. | User’s manual may be difficult for user to understand and confuse. |

Table 10: Software development life cycle

### Phase Detail

#### Phase 1: Requirements Definition

|  |  |  |
| --- | --- | --- |
| Task | Description | Author |
| Identify and clarify system requirements. | Research current systems to collect requirements.  Define main and needed functions the system must include. | Huỳnh Quang Thảo  Trần Thanh Ngoan  Nguyễn Trung Nam  Ngô Tiến Đạt |

Table 11: Requirements definition

#### Phase 2: System and Software Design

|  |  |  |
| --- | --- | --- |
| Task | Description | Author |
| Identify hardware and software requirements. | Find out the suitable hardware and software for the system, as well as its minimum and recommended requirements. | Huỳnh Quang Thảo  Trần Thanh Ngoan  Nguyễn Trung Nam  Ngô Tiến Đạt |
| Decide software architect and clarify software detail design. | - Define the major software components and interfaces.  - Draw core flow diagram, use case diagram, prototype …  - Group meeting to review and modify. | Huỳnh Quang Thảo  Trần Thanh Ngoan  Nguyễn Trung Nam  Ngô Tiến Đạt |
| Design database. | - Design database for the system. | Huỳnh Quang Thảo  Trần Thanh Ngoan  Nguyễn Trung Nam  Ngô Tiến Đạt |

Table 12: System and software design

#### Phase 3: Implementation and Unit Testing

|  |  |  |
| --- | --- | --- |
| Task | Description | Author |
| Implements all functions of system. | Coding all the components. | Huỳnh Quang Thảo  Trần Thanh Ngoan  Nguyễn Trung Nam  Ngô Tiến Đạt |
| Create test plan. | Planning for testing. | Huỳnh Quang Thảo  Trần Thanh Ngoan  Nguyễn Trung Nam  Ngô Tiến Đạt |
| Perform Unit testing | - Write Unit test cases.  - Implement Unit tests. | Huỳnh Quang Thảo  Trần Thanh Ngoan  Nguyễn Trung Nam  Ngô Tiến Đạt |

Table 13: Implementation and unit test

#### Phase 4: Integration and System Testing

|  |  |  |
| --- | --- | --- |
| Task | Description | Author |
| Perform integration test and system test. | - Test groups of modules and test whole the system. | Huỳnh Quang Thảo  Trần Thanh Ngoan  Nguyễn Trung Nam  Ngô Tiến Đạt |

Table 14: Integration and system testing

#### Phase 5: Operation and Maintenance

|  |  |  |
| --- | --- | --- |
| Task | Description | Author |
| Deploy the system | Deploy the system in client environment. | Huỳnh Quang Thảo  Trần Thanh Ngoan  Nguyễn Trung Nam  Ngô Tiến Đạt |
| Create the user’s manuals. | Create a guideline to instruct users using system. | Huỳnh Quang Thảo  Trần Thanh Ngoan  Nguyễn Trung Nam  Ngô Tiến Đạt |
| Do routine maintenance activities. | Do routine maintenance activities for client system. | Huỳnh Quang Thảo  Trần Thanh Ngoan  Nguyễn Trung Nam  Ngô Tiến Đạt |

Table 15: Operation and maintenance

# **Software Requirement Specification**

## User Requirement Specification

### Customer requirement

Customer is user who uses mobile application and wear application and use web services for searching bus route or motorbike route. The customer can use some following functions:

* Mobile application includes:
* Search function includes:
  + - Search arbitrary location map.
    - Search bus route go through two points to four points.
    - Search bus route go through two points to four points with optimization.
    - Search motorbike route go through two points to four points.
    - Search motorbike route go through two points to four points with optimization.
  + View current location on map.
  + Routing function includes:
    - Routing for bus route.
    - Routing for motorbike route.
    - Detect wrong route and recommend true direction when participating traffic.
* Configuration
  + Download offline map.
  + Change notify distance for bus and motorbike.
  + Choose priority search option.
* Wear application includes:
* Map function includes:
  + View current location on map.
  + Navigate a bus station on map.
  + Navigate a motorbike route on map.
* Notification function includes:
  + Show notification when near the bus station user should to leave.
  + Show notification when come to other motorbike’s turn in motorbike route.
  + Show warning message when user goes to wrong route.
  + Show recommend message when user from wrong route near again original searched route.

### Staff requirement

Staff is people who works directly with system and can change information of bus route or bus timetable. Staff can use some following functions:

* Bus management function includes:
  + View bus route information.
  + View bus timetable information.
  + Edit bus route information.
  + Edit bus timetable information.
* A notification shows a new bus route or bus timetable information when data change. Notification management function includes:
  + View all system notifications.
  + View a detail notification.
  + Approve or reject all current system notifications. So all bus routes or bus timetable information will be updated or stay same respectively.
  + Approve or reject a detail notification. So that bus route or bus timetable information will be updated or stay same respectively.

## System Requirement Specification

### External Interface Requirement

#### User interface

* The user interface for mobile application and wear application uses Vietnamese language.
* The user interface for staff uses English language.
* Use consistent palette of colors between the text and the background.
* The user interface for web application displays best on 1024x768-screen size.
* The user interface for mobile application displays best on screen size larger than 4’’.
* The user interface for wear application displays best on screen size from 1.65’’.

#### Hardware Interface

* Smartphone with Wifi or 3G, GPS and Bluetooth.
* Wear has Bluetooth and optional 3G.

#### Software Interface

* Web application: work with Firefox (v30 or above), Chromes (v14 or above), Internet Explorer (v10 or above) browse.
* Mobile application: Android operating system (v 4.3 or above).
* Wear application: Android Wear operating system (API 20 or above)

#### Communication Protocol

* Use HTTP protocol 1.1 for communication between the web browser and the web server.
* Use HTTP protocol 1.1 for communication between the mobile application and the web service.

### System Overview Use Case

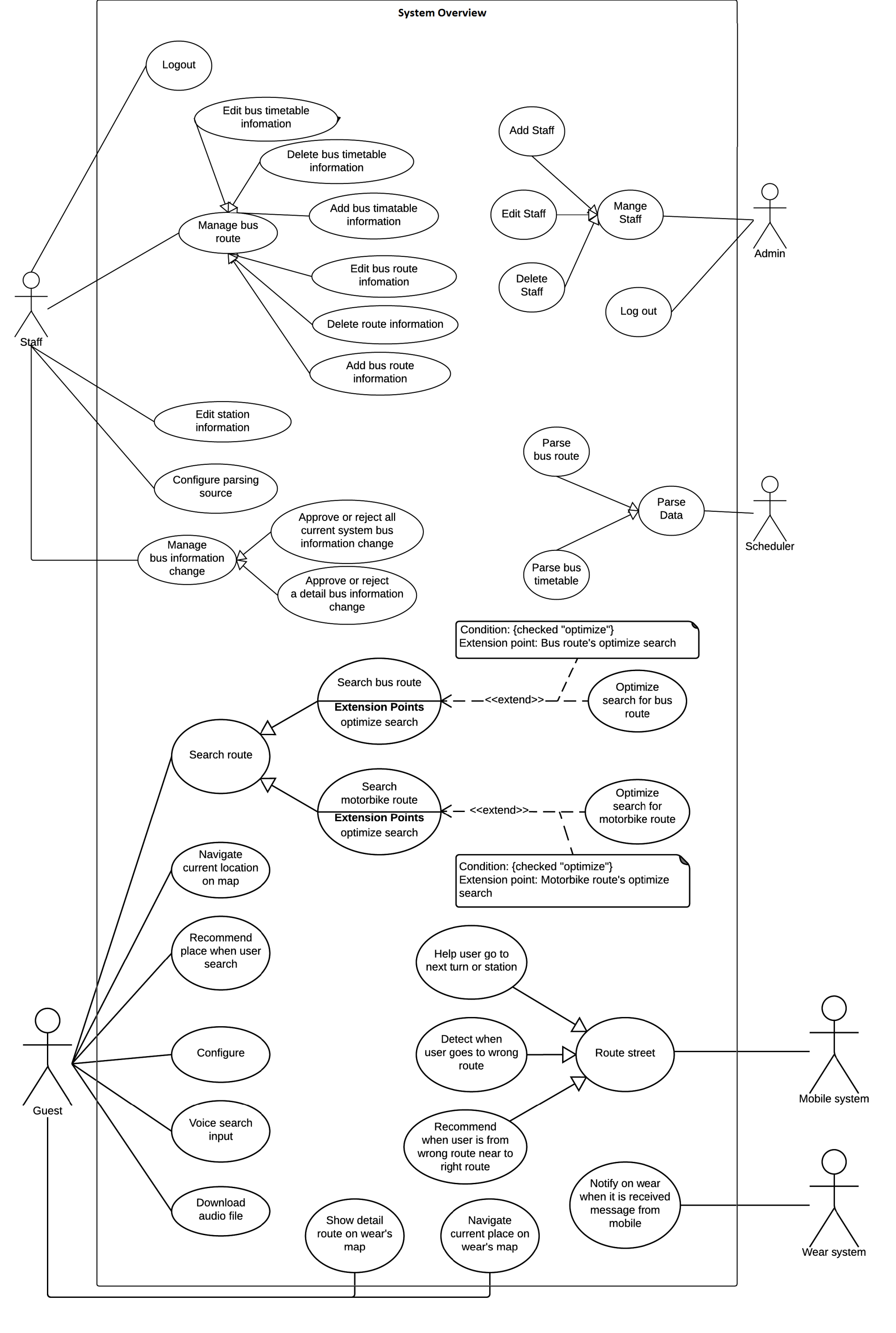


Figure 2: System overview use cas

## Software System Attribute

### Usability

#### Graphic User Interface

For mobile application and wear application, all the texts, labels and alerts will be written in Vietnamese.

For web application, all the texts, labels and alerts will be written in English.

#### Usability

* The system usability is easy to use that will need less than 3 days of training for company staffs to use the system management.
* Customers can use all mobile application’s functions by reading help manual inside mobile application.
* Bus drivers need less than 1 hours of training to use bus driver’s mobile application.

#### Installation

* User can follow installation and manual guide for installation. If there are any problems, user cans contacts developer for help.

### Reliability

* Scheduler task runs at OAM everyday with 100% execution rate.

### Availability

* N/A

### Security

* All data are validated before saving to database.
* Staff password must be encrypted in database.
* All data from background handler or bus driver’s uploaded data must be approved before saving to database.
* All privacy information such as search history is only stored at local database.

### Maintainability

* The system is separated into modules.

### Portability

* Staff can use application on every OS supported web browser.
* Customer can use mobile application on every Android smartphone that have version greater than 4.3 and wear that have version from API 20.

### Performance

* Requests from mobile application to server for finding bus route are responded in less than 15 seconds at network connection 8 Mbps.
* Algorithm for finding bus route must run less than 10 seconds for Ho Chi Minh bus system.
* Mobile application synchronizes data with server in less than 1 minute at network connection 8 Mbps.

## Conceptual Diagram

Web conceptual diagram:

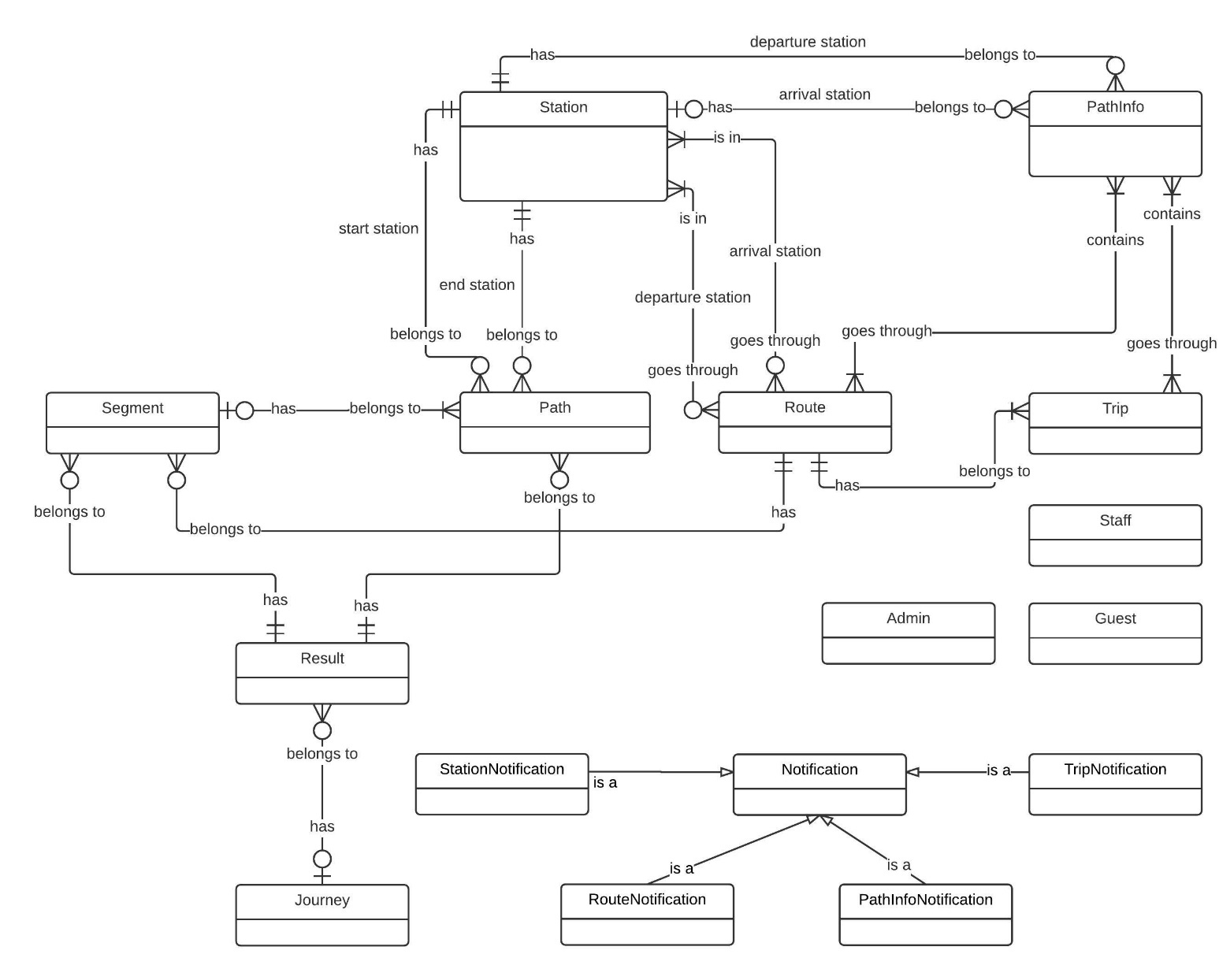


Figure 43: Conceptual Diagram

4.2. Mobile conceptual diagram:

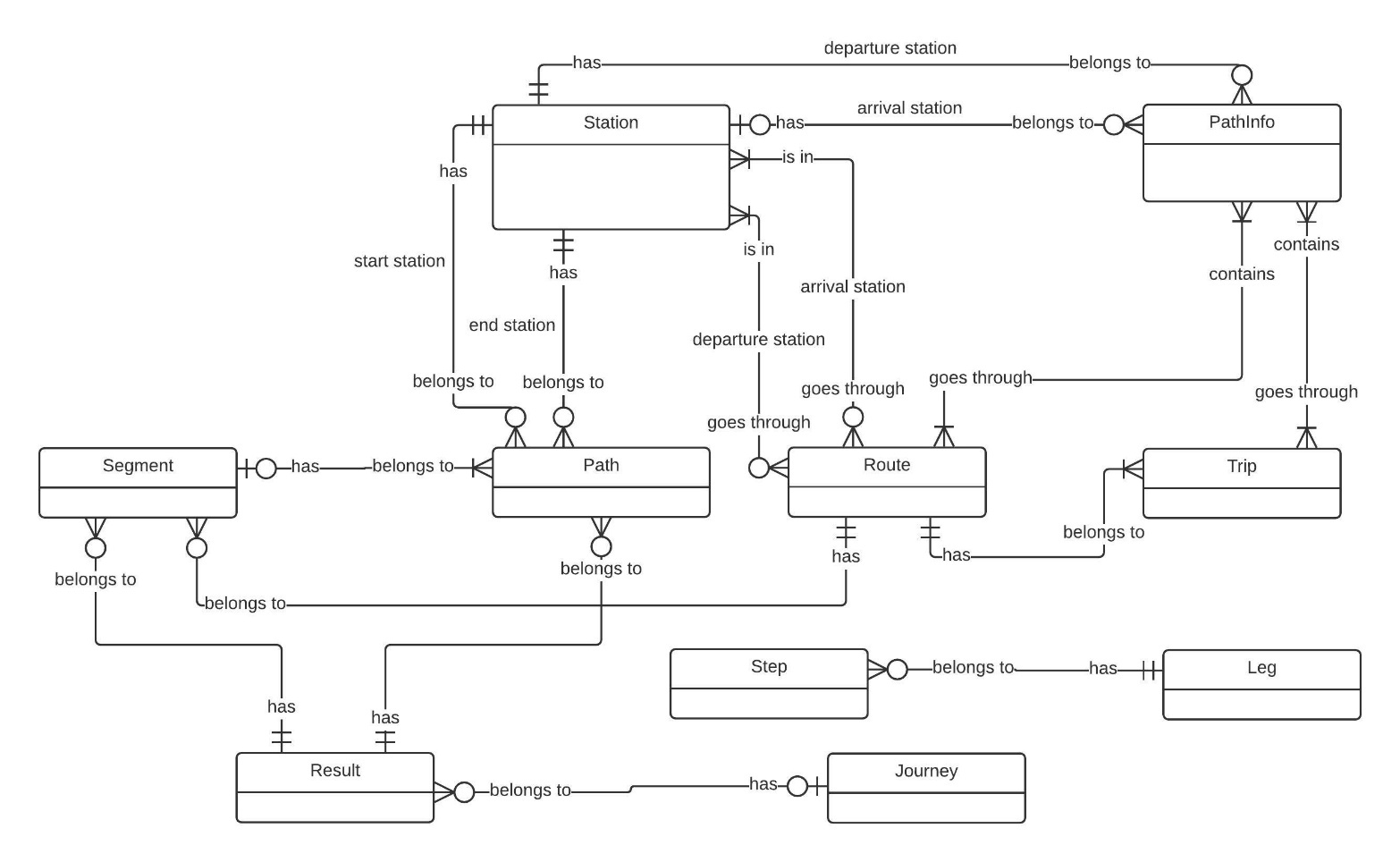


Figure 44: Mobile Conceptual diagram

# **Software Design Description**

## Design Overview

* This document describes the technical and user interface design of SWR system. It includes the architectural design, the detailed design of common functions and business functions and the design of database model.
* The architectural design describes the overall architecture of the system and the architecture of each main component and subsystem.
* The detailed design describes static and dynamic structure for each component and functions. It includes class diagrams, class explanations and sequence diagrams for each use cases.
* The database design describes the relationships between entities and details of each entity.
* Document overview:
  + Section 2: gives an overall description of the system architecture design.
  + Section 3: gives component diagrams that describe the connection and integration of the system.
  + Section 4: gives the detail design description, which includes class diagram, class explanation, and sequence diagram to details the application functions.
  + Section 5: describe a fully attributed Entity Relationship Diagram.

## System Architecture Design

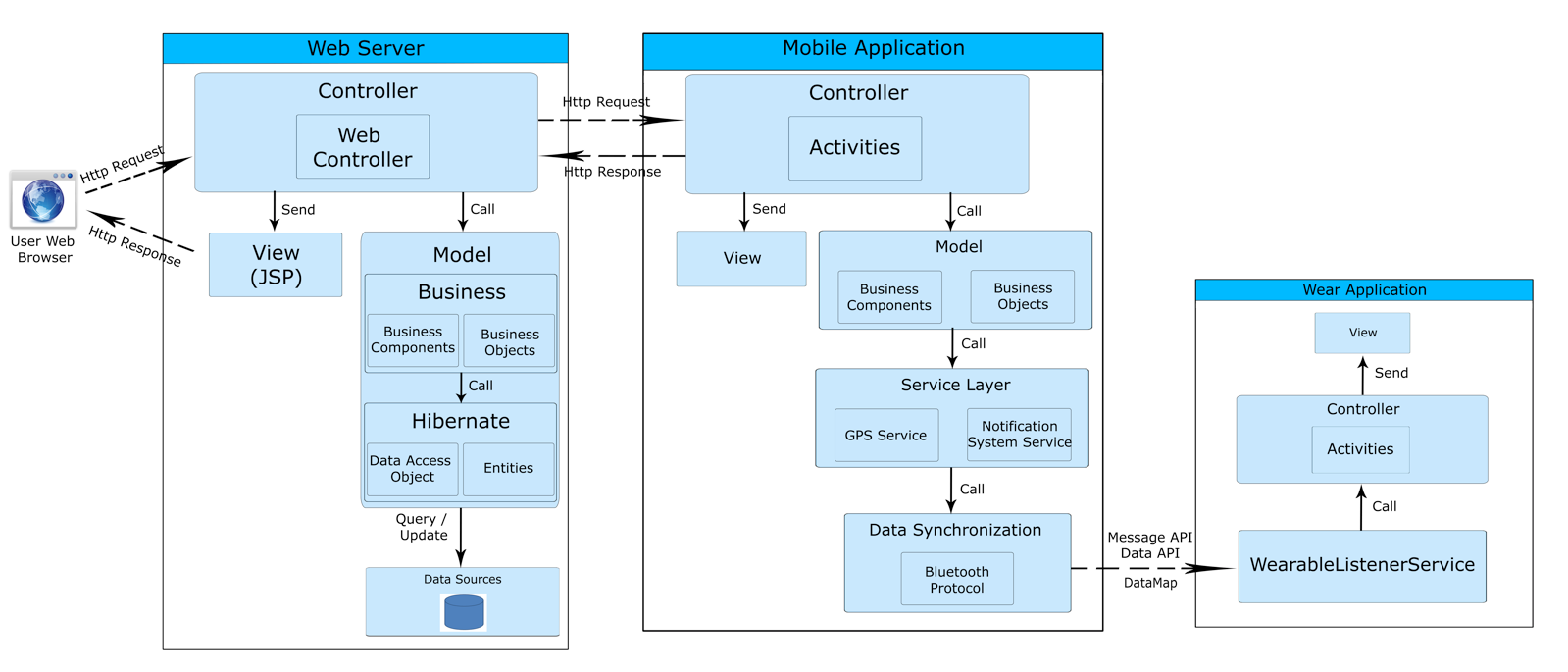


Figure 45 System architecture design

## Component Diagram

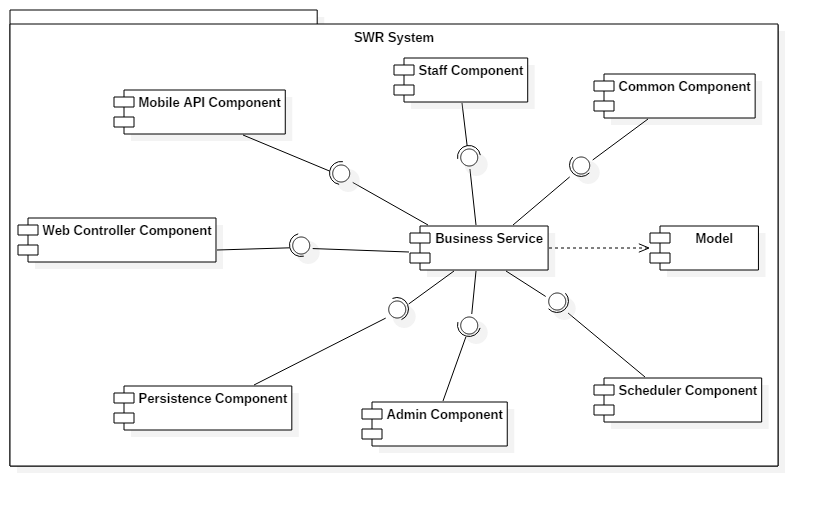


Figure 46 Component Diagram

## Detailed Description Class Diagram

### Web System class diagram

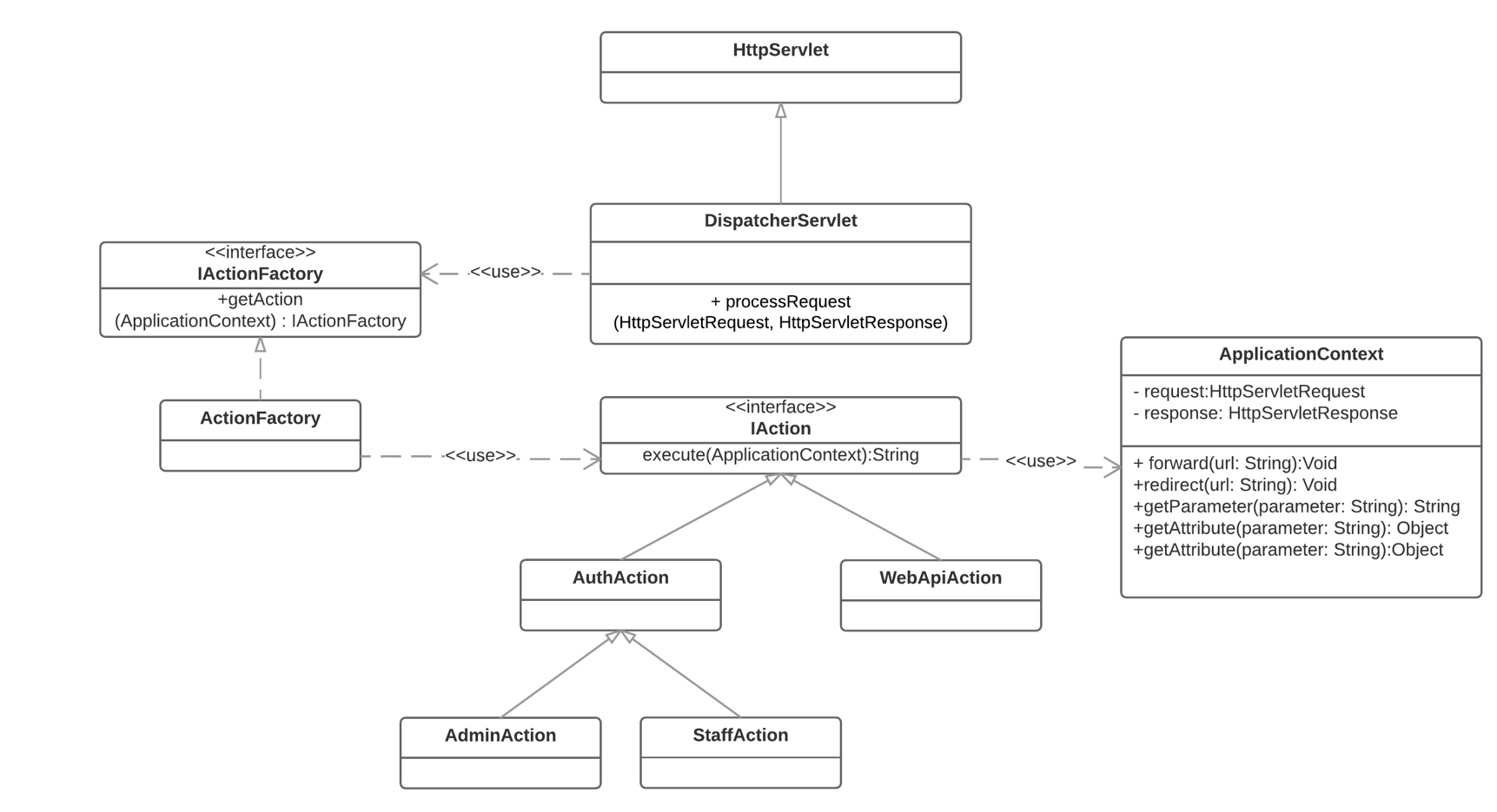


Figure 47: Web system class diagram

### Web class diagram

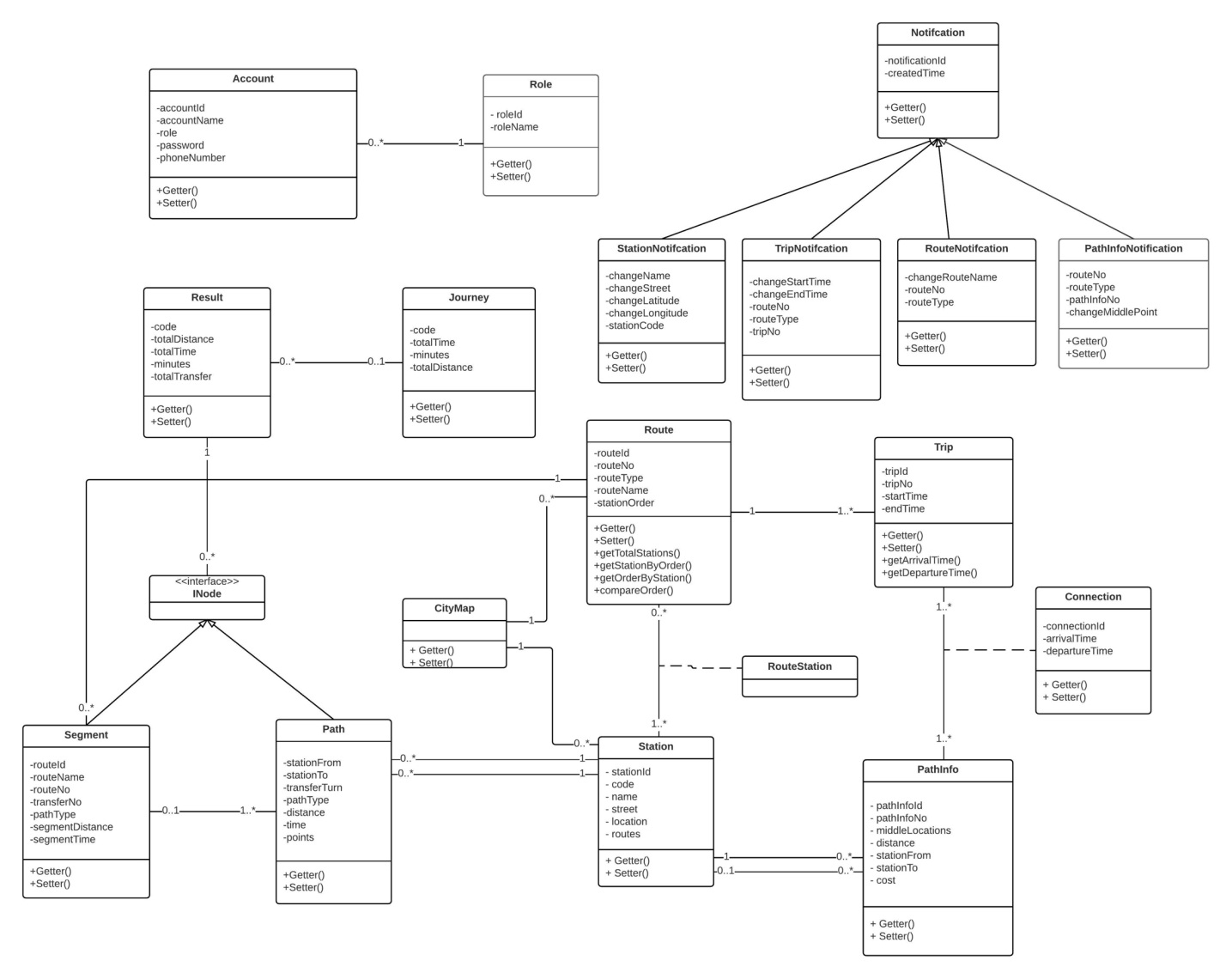


Table 56 Class dictionaryInterface

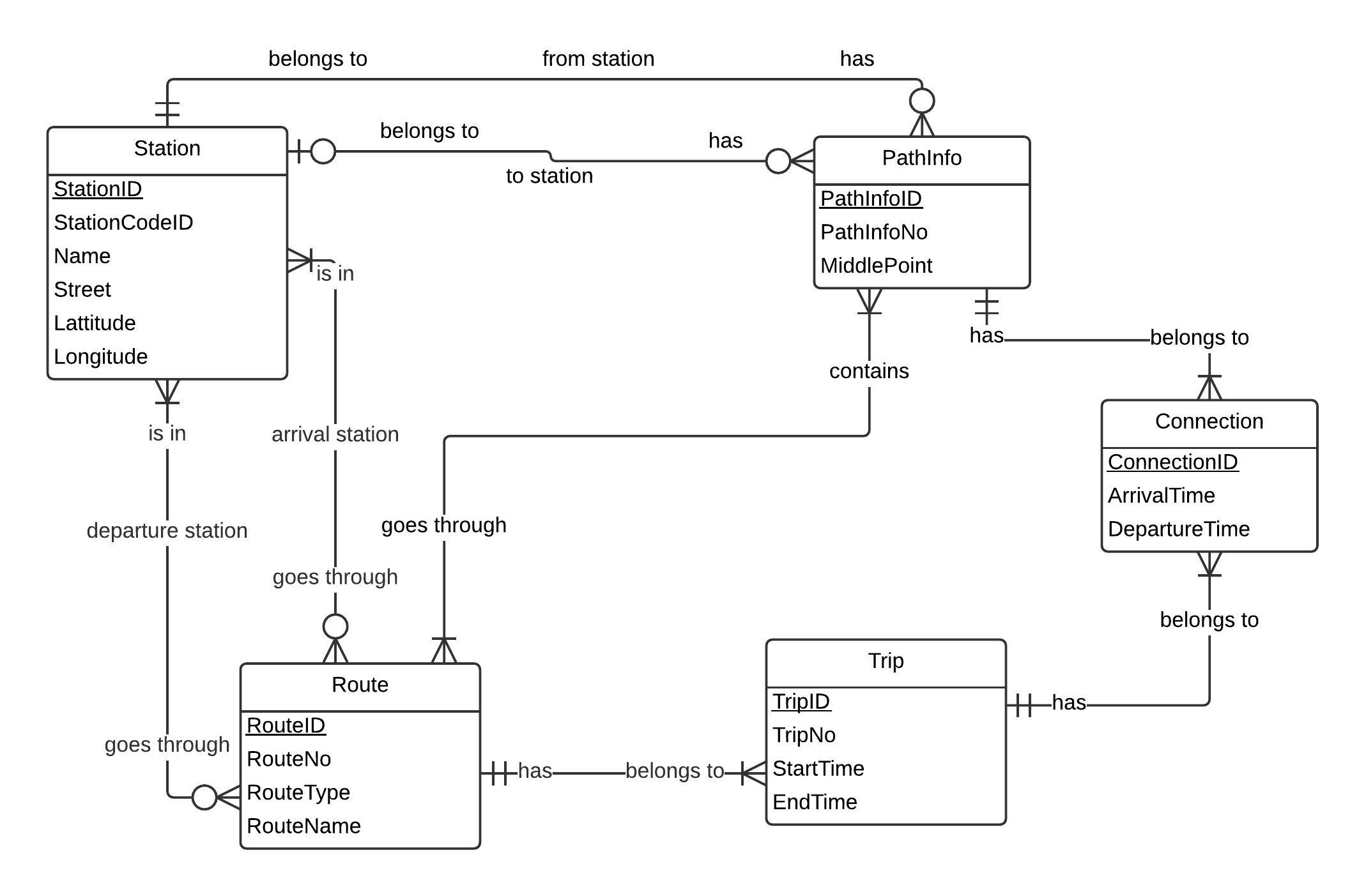
### Component interface

#### Web service interface

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| © | Description | Input | Output | Output Format | Exception |
| /api/bus/twopoint | Find bus route between two points | **latStart:double**. Latitude of start location.  **longStart:double**.Longitude of start location.  **latEnd:double**. Latitude of end location.  **longEnd:double**. Longitude of end location.  **addressStart:String**. Address of start location.  **addressEnd:String**. Address of end location.  **hour:Integer**. Hour that customer wants to search.  **minute**: minute that customer wants to search.  **walkingDistance:Integer**. Walking distance.  **transferTurn:Integer**. Maximum transfers for one search. | Json string of Result object. | Json | ParameterFormatException |
| api/bus/multipoint | Find bus route from three point to four point with optimize or not | **latStart:double**. Latitude of start location.  **longStart:double**.Longitude of start location.  **latEnd:double**. Latitude of end location.  **longEnd:double**.Longitude of end location.  **latMidFirst:double**. Latitude of first middle point.  **longMidFirst:double**.Longitude of first middle point.  **latMidSecond:double**. Latitude of second middle location.  **longMidSecond:double**.Longitude of second middle location.  **addressStart:String**. Address of start location.  **addressEnd:String**. Address of end location.  **addressMidFirst:String**. Address of first middle location.  **addressMidSecond:String**. Address of second middle location.  **isOp:Boolean**. Marking this search is optimize or not.  **hour:Integer**. Hour that customer wants to search.  **minute**: minute that customer wants to search.  **walkingDistance:Integer**. Walking distance.  **transferTurn:Integer**. Maximum transfers for one search. | Json String of Journey object. | json | ParameterFormatException |

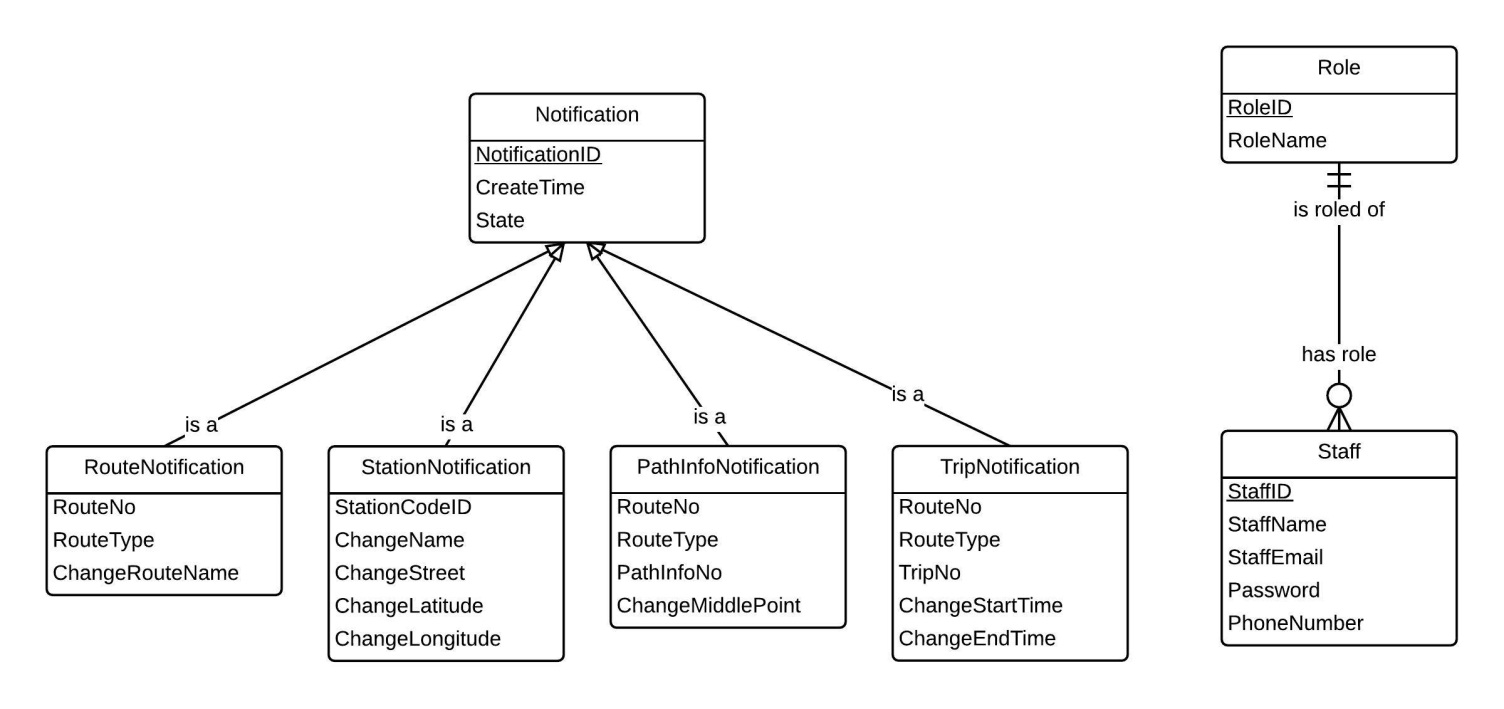
## Database design

### Main Entity Relationship diagram



### 6.2. Sub entity Relationship diagram

#### 6.2.1. Main entity relationship diagram



## Algorithm

### Raptor algorithm

#### Problem definition:

We need to find the fasted path (the path with less time for traveling) from one station to one station go through some bus routes with some additional conditions: departure time, number of transfers.

#### Introduction:

This problem can be solved by using Raptor algorithm. Raptor algorithm is an algorithm based on paper named “Round-Based Public Transit Routing”. This paper is published by researchers Daniel Delling, Thomas Pajor and Renato F. Werneck on 2012.

Above problem can be known as computing all Pareto-optimal journeys in a dynamic public transit network for multiple criteria such as departure time and number of transfers.

Existing algorithms consider this as a graph problem, and solve it using variants of Dijkstra algorithm such as Layered Dijkstra (2004), multi-label-correcting algorithm (2008).

Starting from arrival time and number of transfers as criteria, this algorithm can be extended to handle more arbitrary additional criteria such as arrival time, ticket price, change time at each station or reliability of transfers.

#### Attribute Definition:

In below algorithm explanation, we use some notations:

* maximum bus transfers for this search. We can choose this dynamically when searching.
* is set of city’s stations.
* is set of city’s routes.
* is set of trips of all routes. And is set of trips of route . In our algorithm, we will sort those trips of each route based on departure time of each trip for searching optimization.
* earliest arrival time that we can come to this station using at most k transfers. Our algorithm will optimize those set of values. Note that two points:

1. This time is real time base on departure time (ie: 5:30 PM, 8:15 AM)
2. The total transfers can be any from to . If value equals 0, it means we cannot reach this station on time by using any routes.

* earliest arrival time to this station. After loops finish,
* be the earliest trip in route r that one can catch at station
* be the departure time of trip t at . Note that, currently, our bus system, arrival time and departure time of trip t at are equals. That means no wait time at each station of each trip.
* be the arrival time of trip t at . Note that, currently, our bus system, arrival time and departure time of trip t at are equals. That means no wait time at each station of each trip.

We also have some variables which means:

* **Q**: a map. This map has key is a route and value is a station in this route which previous round has been optimized.
* **TraceUsedRoute[i][j] = R** means: Best way come to j-station using at most i transfers using route R.
* **TraceFromStation[i][j] = S** means: best way come to j-station using at most i transfers must go on at station S.
* **Result[i][j] = T** means: best way come to j-station using at most i transfers it T times.

#### Solution:

Method signature: **Raptor**(source s, target t, departure time r, maximum transfer K)

Step 1: initialize all variables Q, TraceUsedRoute, TraceFromStation, Result

Step 2: optimize each stations for total K rounds. After K rounds, we can have best time go to each stations at maximum K transfers.

**For all** k 🡨 1,2,3,…K **do**

Step 3: find all previous round optimize stations. Find all routes go through this station.

**For all** marked station p **do**

**For all** routes r go through station p **do**

**If** (r, p’) in Q for some p’ **then**

**If** r.order(p) < r.order(p’) **then**

Change (r, p’) -> (r,p) in Q

**Endif**

**Else**

Add (r, p’) 🡪 Q

**Endif**

**endfor**

**endfor**

Step 4: travel again each route from marked station and optimize time at each station.

**For all** pair (r,p) in Q **do**

Trip t = null

**Forall** stop beginning with p in route r **do**

**If** t != null **and** < **then**

=

Add to marked stations for next round

Step 5: save to trace array for print result later, if necessary

traceFromStation[k][ = p

traceUsedRoute[k][ = r

result[k][ = t

**endif**

**if**  **then**

**endif**

**endfor**

step 6. Check if no optimize stations anymore

**if** no marked station in this turn **then**

**break;**

**endif**

**endfor**

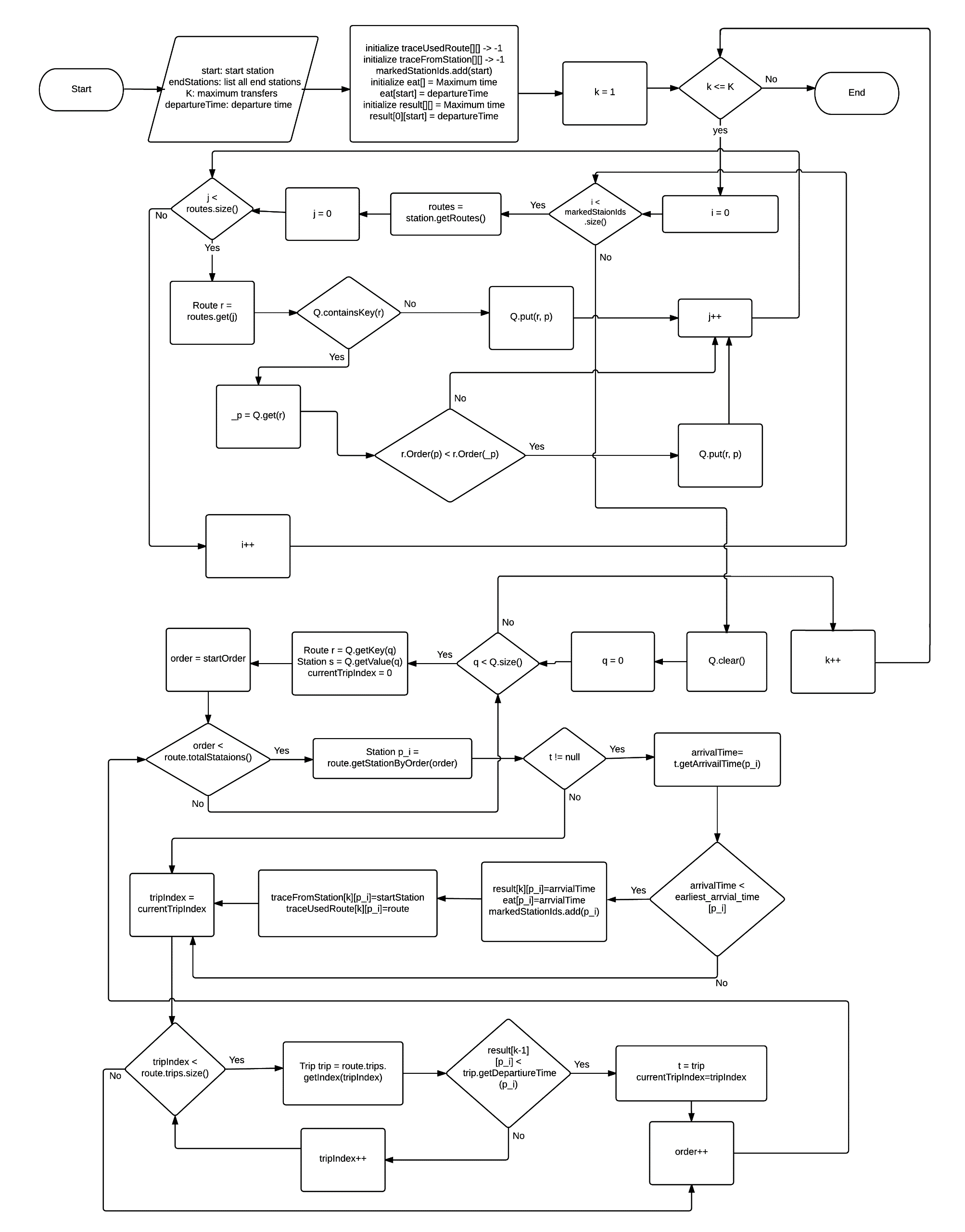
#### Algorithm complexity

The worst-case running time of our algorithm can be bounded as follows:

In every round, algorithm scans each route at most once. If is the number of stations of route , then we look at stations in total to process the route. For each station , we must find the earliest trip. If we keep the list of trips serving r sorted by time, while traversing r we can find all values with a single sweep over this list, since can only decrease. In total, RAPTOR takes time for each round.

If we call sum of all stations and all trips (of all routes) in city equals to . Total algorithm complexity is: )

#### Flowchart



### Search bus two point:

#### Problem definition

* User request 2 points, start point and end point and their settings includes: walking distance, transfer turn. System get requested points and settings, then search for suitable routes, for user travel from start point to end point.

#### Attribute definition

* City Map: includes list of routes and list of stations:
* Each station in list of stations includes attributes: station’s name represent name of station, station’s code represent code of station, station’s street represent where the station locate, station’s latitude and longitude for coordinating station on the map.
  + Each route in list of routes includes attributes: route’s name, route’s no, list of paths, list of trips.
    - List of paths: each path in list of paths include attributes: path no for represent the ordered station of route, station in from and station …,
    - List of trips: each trip in list of trips represent start time and end time of route.
* List of stations round by start point: called LIST\_START\_STATIONS represent stations are found round by start point.
* List of stations round by end point: called LIST\_END\_STATIONS represent stations are found round by end point.
* Start path: represent the path includes calculated distance, time from one station in LIST\_START\_STATIONS to start point.
* End path: represent the path include calculated distance, time from one station in LIST\_END\_STATIONS to end point.
* List of results: called LIST\_RESULTS represent the list result is returned from Raptor’s algorithm and each result in this list will show to user.

#### Solution

* Search route algorithm is based on Raptor’s algorithm.
* To solve problem, we follow these steps:
  + Step 1 - Calculate distance between two stations: if distance less than walking distance, system will show message for user can walking between two points.
  + Step 2 - Find stations nearby selected points: Iterate list of stations of city map, find station have a distance to selected point less than walking distance, then the suitable station will be added into list of stations round by selected point. Within two selected point, we have two output: LIST\_START\_STATIONS and LIST\_END\_STATIONS.
  + Step 3 – LIST\_START\_STATIONS, we build start path from one station in list to start point.
  + Step 4 – LIST\_END\_STATIONS, we build end path from one station in list to end point.
  + Step 5 – Calculating total distance and total time with entries are start path, start point, end point and end path by using Raptor’s algorithm (reference 7.1 Raptor algorithm). Each result is returned from Raptor’s algorithm will be added into list of results, we call it is LIST\_RESULTS:
    - Total distance is total of three distance below:
      * Distance of start path.
      * Distance of end path.
      * Distance from one station in LIST\_START\_STATIONS to one station in LIST\_END\_STATIONS.
    - Total time is total of three time below:
      * Time of start path.
      * Time of end path.
      * Time from one station in LIST\_START\_STATIONS to one station in LIST\_END\_STATIONS.
  + Step 6 – If LIST\_RESULTS is empty, we show message for notify user know no route found from start point to end point. If LIST\_RESULT is not empty, we sort the list follow three priorities below:
    - The total transfer turn is high priority: we sort list base on increasing transfer turn.
    - The total time is medium priority: we sort list base on increasing time.
    - The total distance is low priority: we sort list base on increasing total distance.
  + Step 7 – With LIST\_RESULTS sorted, we get the first three element for showing to user.

#### Algorithm complexity

In two point’s algorithm, each start point will call Raptor’s algorithm once. Two point’s algorithm get maximum 10 start point so Raptor’s algorithm is called maximum 10 times. So that, two points algorithm complexity is O(K\*N): if we call sum of all stations and all trips (of all routes) in city equals to

#### Flowchart

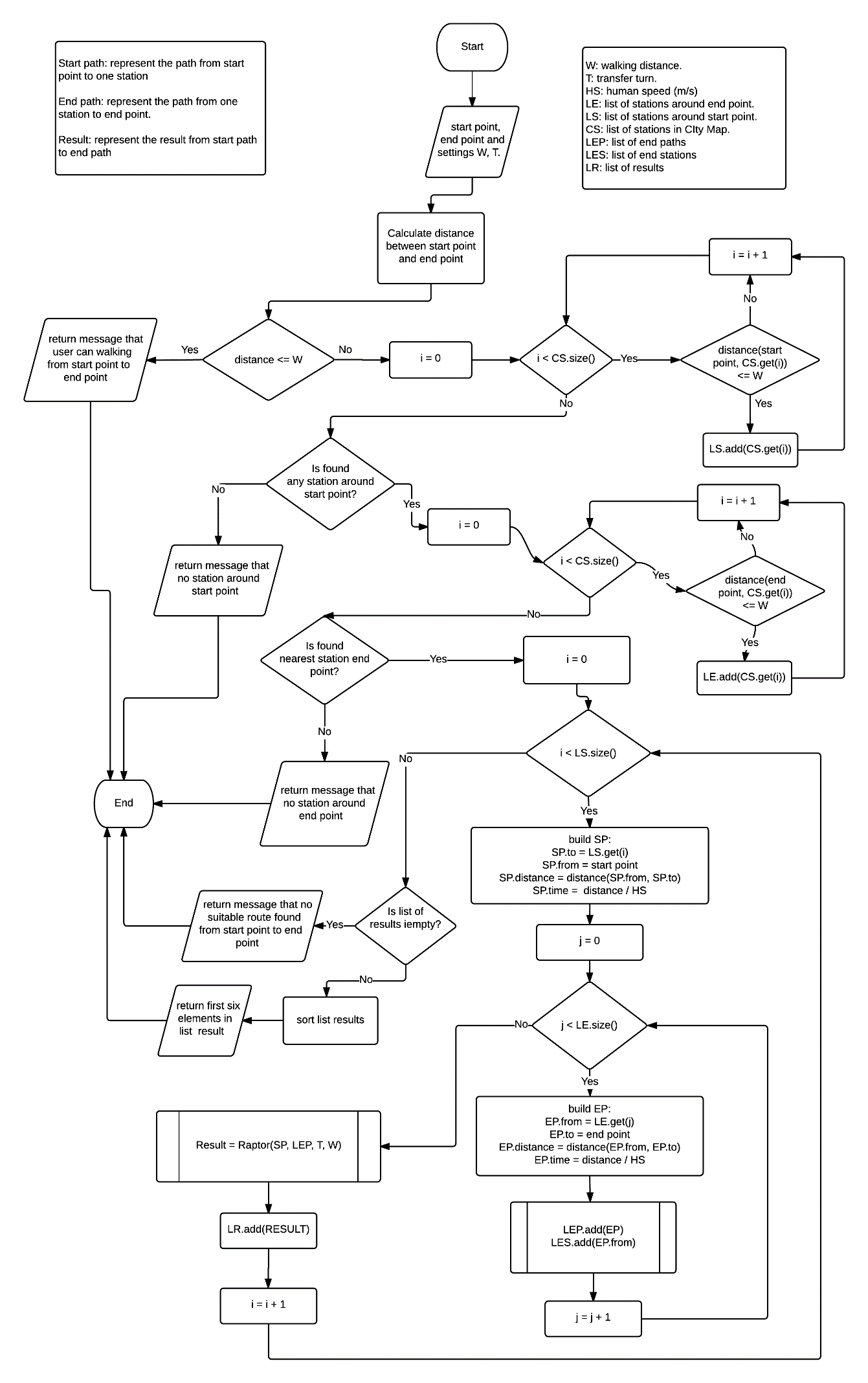


Figure 81: Search bus two point algorithm – flowchart

### Search bus more than two points

#### 7.3.1. Problem definition

**-** User request four points, start point, end point, two middle points and their settings includes: walking distance, transfer turn and not optimize option. System get requested points and settings, then search for suitable routes, for user travel from start point to end point though two middle points.

#### 7.3.2. Attribute definition

* COMBINATIONS: represent list of element which combination from two point adjacent.
* Journey: represent the journey from start point to end point though middle points, includes: total distance, time.

#### 7.3.3. Solution

* Multi points search route base on “Search bus two point algorithm”.
* To solve problem, we follow these steps:
  + Step 1: Swap two point of middle points, then we have two results represent list of point will be passed. Suppose four points are A, B, C, D so we have 2 results are ABCD, ACBD.
  + Step 2: With two result above. We exhaust two points of four points of each result, each point and the next point will be combined to an element. And each combined element will be added into list of element we call it is COMBINATIONS. Suppose four inputted points from user are A, B, C, D, so COMBINATIONS is AB, BC, and CD.
  + Step 3: We use “two points algorithm” to solve each element of COMBINATIONS. Each element is include LIST\_RESULTS – the attribute from algorithm.
  + Step 4: Build list of journey, we call it is LIST\_JOURNEYS. Each journey will be built from COMBINATIONS and includes:
    - Total time: total of time from each result of LIST\_RESULT of each element in COMBINATIONS.
    - Total distance: total of distance from each result of LIST\_RESULT each element of COMBINATIONS.
  + Step 5: Sort LIST\_JOURNEYS follow two priorities:
    - Time is high priority. We sort the list base on increasing time.
    - Distance is low priority. We sort the list base on increasing distance.
  + Step 6: With LIST\_JOURNEYS sorted, we get first six element for show to user.

#### 7.3.4. Algorithm complexity

Search bus more than two points will call “Two point’s algorithm” maximum 3 times. So that, search bus more than two points have algorithm complexity is O(K\*N): if we call sum of all stations and all trips (of all routes) in city equals to .

#### 7.3.5. Flowchart

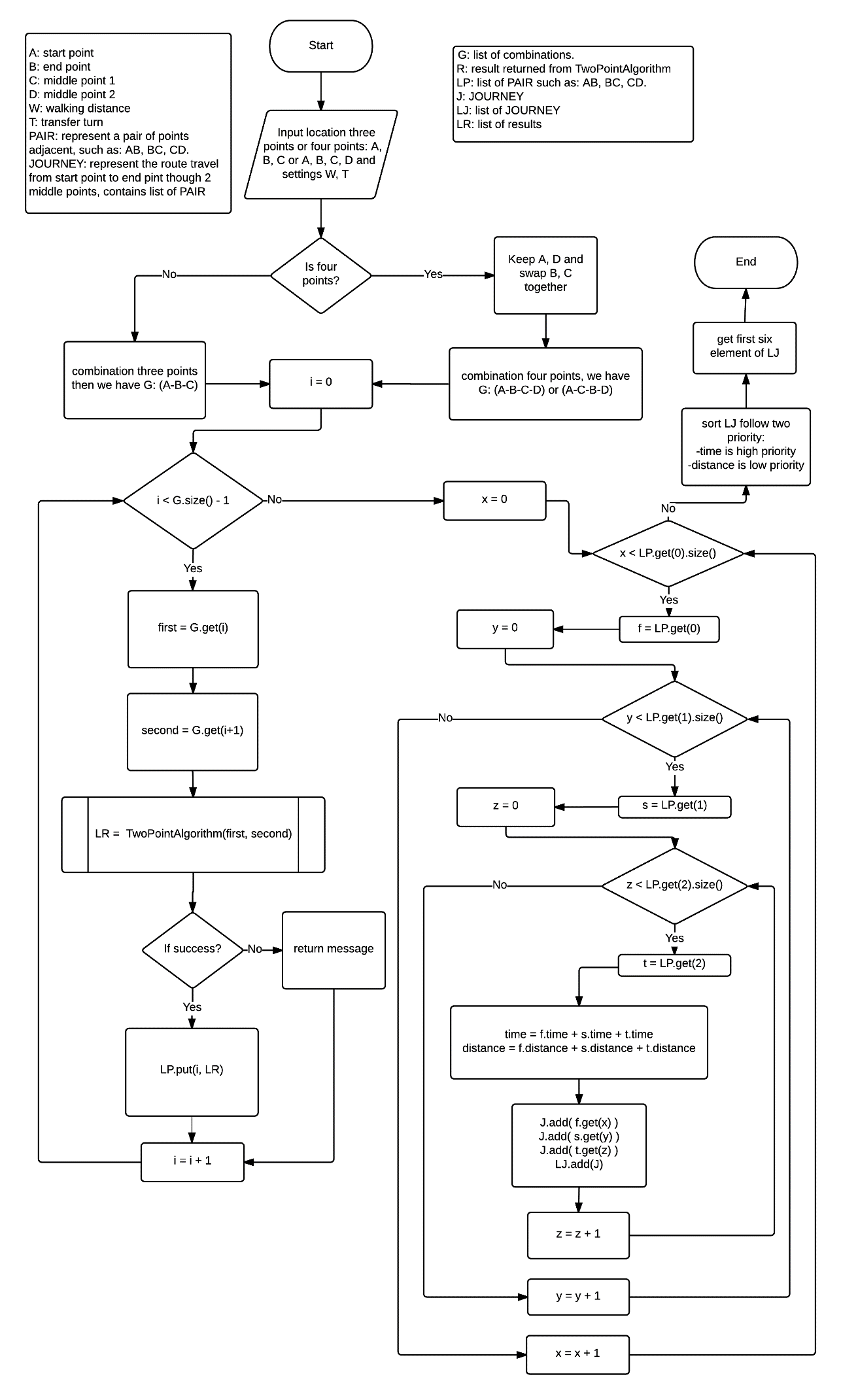


Figure 82: Search bus more than two points algorithm – flowchart

### Search bus more than two points with optimize

#### Problem definition

* User request four points, start point, end point, two middle points and their settings includes: walking distance, transfer turn and optimize option. System get requested points and settings, then search for suitable routes, for user travel from start point to end point though two middle points.

#### Introduction

#### Attribute definition

* LIST\_JOURNEYS (reference attribute LIST\_JOURNEYS in search bus more than two points)
* COMBINATIONS: represent the list of element which combination from four points.

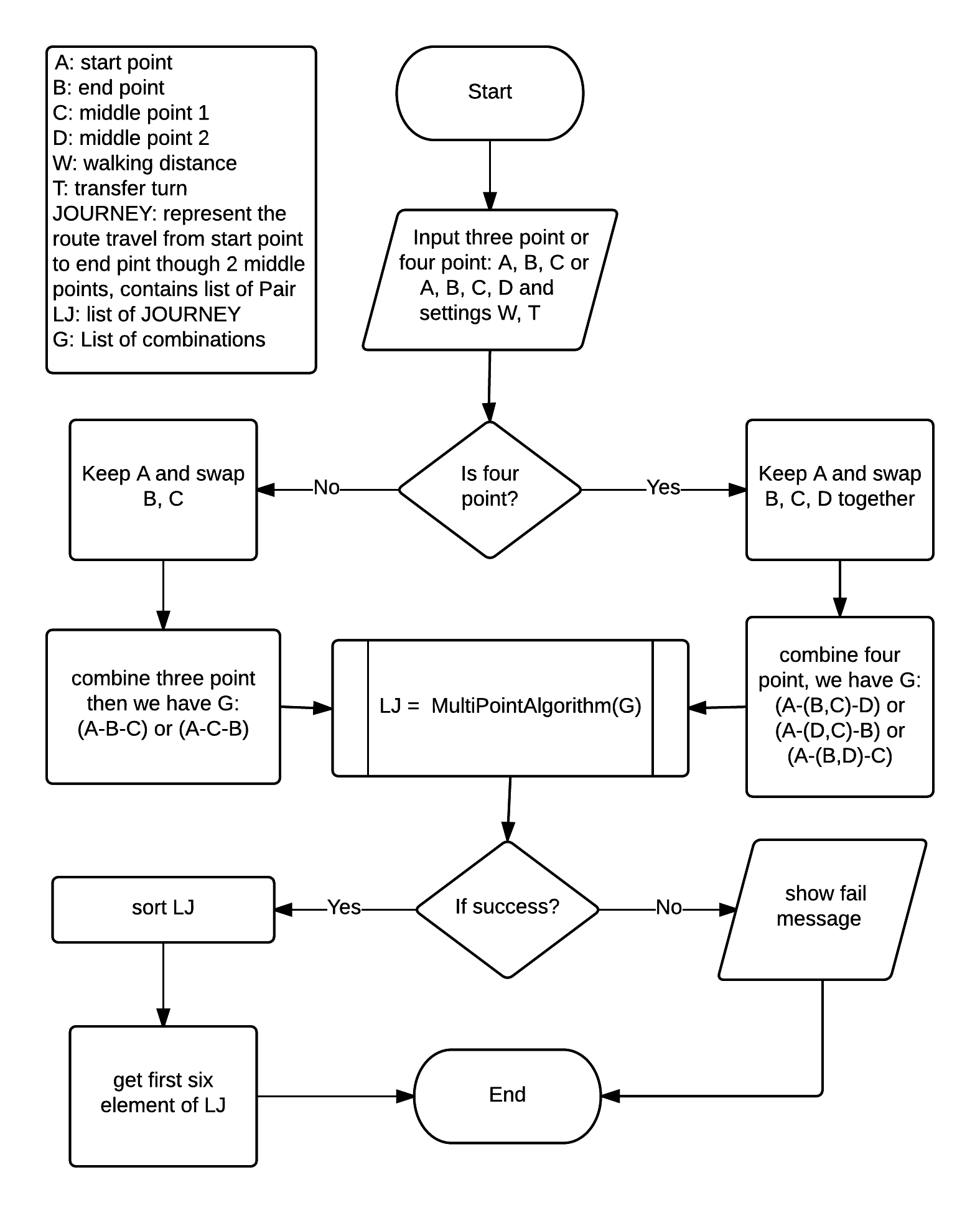
#### Solution

* Multi points search optimize route base one “Multi points algorithm not optimize”.
* To solve the problem, we follow these steps:
  + Step 1: Within four points: start point, end point and two middle points. We keep the start point and swap the remaining three points for creating the combinations, well call it is COMBINATIONS which include list of elements combination from four points. Suppose four inputted point from user is A, B, C, D so we have the COMBINATIONS is ABCD, ACDB, and ABDC.
  + Step 2: We use “multi point algorithm not optimize” to solve for each element in COMBINATIONS.
  + Step 3: With all LIST\_JOURNEYS are returned from “multi point algorithm not optimize” will retrieved first six element after the list is sorted follow two priorities below:
    - Time is high priority. We sort the list base on increasing time.
    - Distance is high priority. We sort the list base on increasing distance.

#### Algorithm complexity

Search bus more than two points with optimize will call “search bus more than two points” algorithm maximum 3 times. So that, the algorithm complexity is O(K\*N): if we call sum of all stations and all trips (of all routes) in city equals to .

#### Flowchart



### Search motorbike more than two points

#### Problem definition

* User request search more than two points for motorbike no optimize including start point, end point, and two middle points and with some additional conditions: departure time, number of transfers. System gets requested points and settings, then send request to Google Map Direction API (The Google Maps Directions API is a service that calculates directions between locations using an HTTP request) to calculates directions between locations, for user travel from start point to end point though two middle points.

#### Attribute definition

* Reference: <https://developers.google.com/maps/documentation/directions/intro>
* COMBINATIONS: represent list of element which combination of points together.
* PAIRS: represent list of element which combination from two points adjacent.
* Routes: Each element of the routes array contains a single result from the specified origin and destination. Routes consist of nested [Legs](https://developers.google.com/maps/documentation/directions/intro#Legs) and [Steps](https://developers.google.com/maps/documentation/directions/intro#Steps).
* Each element in the legs array specifies a single leg of the journey from the origin to the destination in the calculated route.
* A step is the most atomic unit of a direction's route, containing a single step describing a specific, single instruction on the journey.
* geocoded\_waypoints contains an array with details about the geocoding of origin, destination and waypoints.
* Status contains metadata on the request.

#### Solution

* Multi points search route base on “Search Two points with Google Map Direction API”.
* To solve problem, we follow these steps:
  + Step 1: Within four points: start point, end point and two middle points. We keep the start point and end location then swap the remaining two points for creating the combinations, well call it is COMBINATIONS which include list of elements combination from three or four points. Suppose we assign as follows: start location = A, end location = B, middle location 1 = C and middle location 2 = D.

With three inputted point from user is A, B, C so we have the COMBINATIONS is ABC.

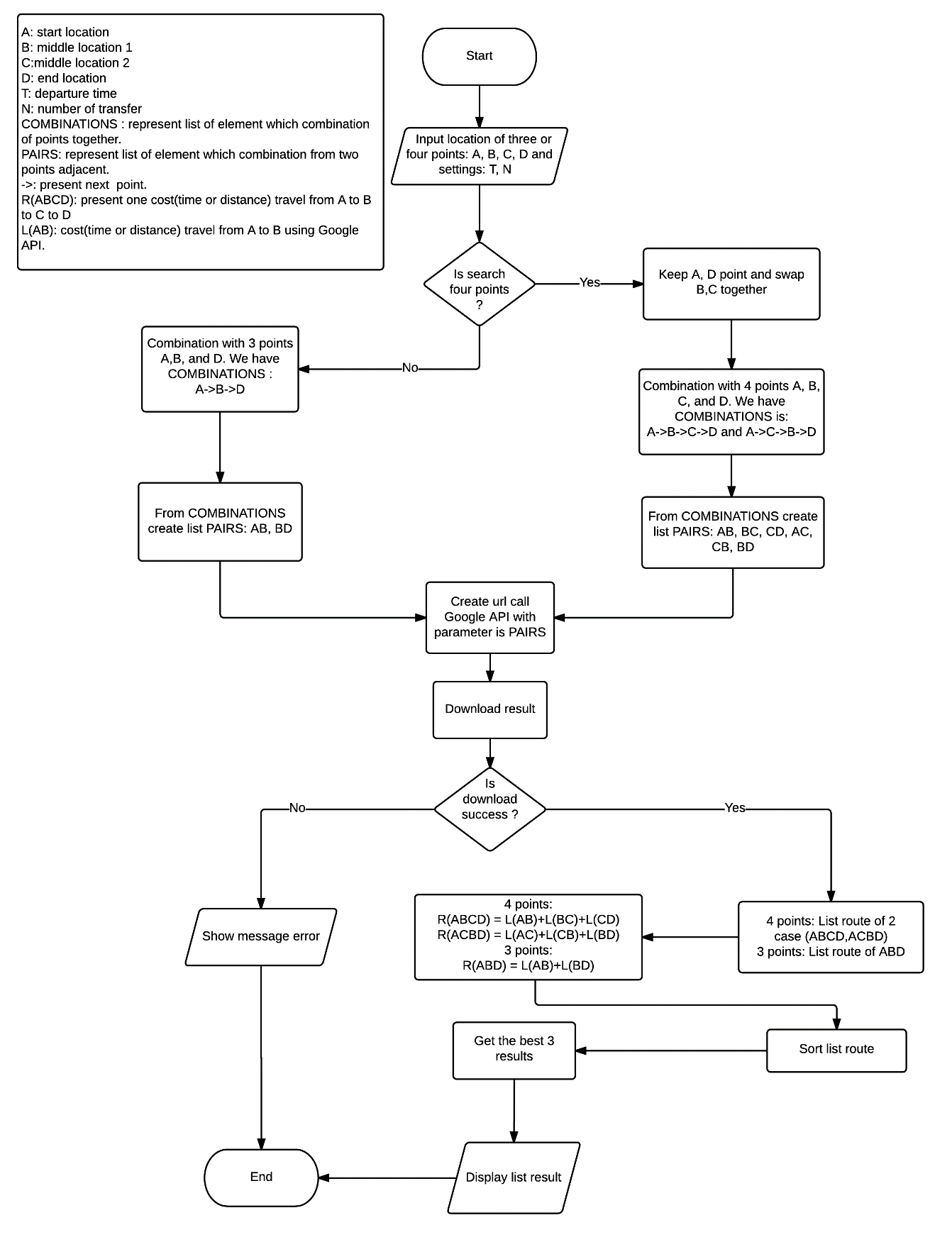
With four inputted point from user is A, B, C, D so we have the COMBINATIONS is ABCD, ACBD.

* + Step 2: We exhaust two points from COMBINATIONS, each point and the next point will be combined to an element. And each combined element will be added into list, we call it is PAIRS.
    - With COMBINATIONS is ABC, we have list PAIRS is AB, BC.
    - With COMBINATIONS is ABCD, ACBD, we have list PAIRS is AB, BC, CD, AC, CB, BD.
  + Step 3: We use “search two point with Google Map Direction API” to solve each element of PAIRS. When the Google Maps Directions API returns results, it places them within a (JSON) routes array. Even if the service returns no results (such as if the origin and/or destination doesn't exist) it still returns an empty routes array.
  + Step 4: Parse JSON into Object, we have list Leg.
  + Step 5: We combination list Leg together to create list result present for search three or four points.
  + Step 6: The list result is sorted follow two priorities below:
    - Time is high priority. We sort the list base on increasing time.
    - Distance is high priority. We sort the list base on increasing distance.
  + Step 7: Get 3 results the best.

#### Algorithm complexity

* O(K): K is total number of requests had send to Google API.
* Three point: K = 2.
* Four point: K = 6.

#### Flowchart



### Search motorbike more than two points optimize

#### Problem definition

* User request search more than two points for motorbike optimize including start point, end point, two middle points and with some additional conditions: departure time, number of transfers and optimize option. System gets requested points and settings, then send request to Google Map Direction API (The Google Maps Directions API is a service that calculates directions between locations using an HTTP request) to calculates directions between locations, for user travel from start point to end point though two middle points.

#### Attribute definition

* Reference: <https://developers.google.com/maps/documentation/directions/intro>
* COMBINATIONS: represent list of element which combination of list points together.
* Routes: Each element of the routes array contains a single result from the specified origin and destination. Routes consist of nested [Legs](https://developers.google.com/maps/documentation/directions/intro#Legs) and [Steps](https://developers.google.com/maps/documentation/directions/intro#Steps).
* Each element in the legs array specifies a single leg of the journey from the origin to the destination in the calculated route.
* A step is the most atomic unit of a direction's route, containing a single step describing a specific, single instruction on the journey.
* geocoded\_waypoints contains an array with details about the geocoding of origin, destination and waypoints.
* Status contains metadata on the request.

#### Solution

* Multi points search optimize route base one “Google Map Direction API”.
* To solve the problem, we follow these steps:
  + Step 1: Within four points: start point, end point and two middle points. We keep the start point and swap the remaining three points for creating the combinations, well call it is COMBINATIONS which include list of elements combination from three or four points. Suppose we assign as follows: start location = A, end location = B, middle location 1 = C and middle location 2 = D.

With three inputted point from user is A, B, C so we have the COMBINATIONS is ABC, ACB.

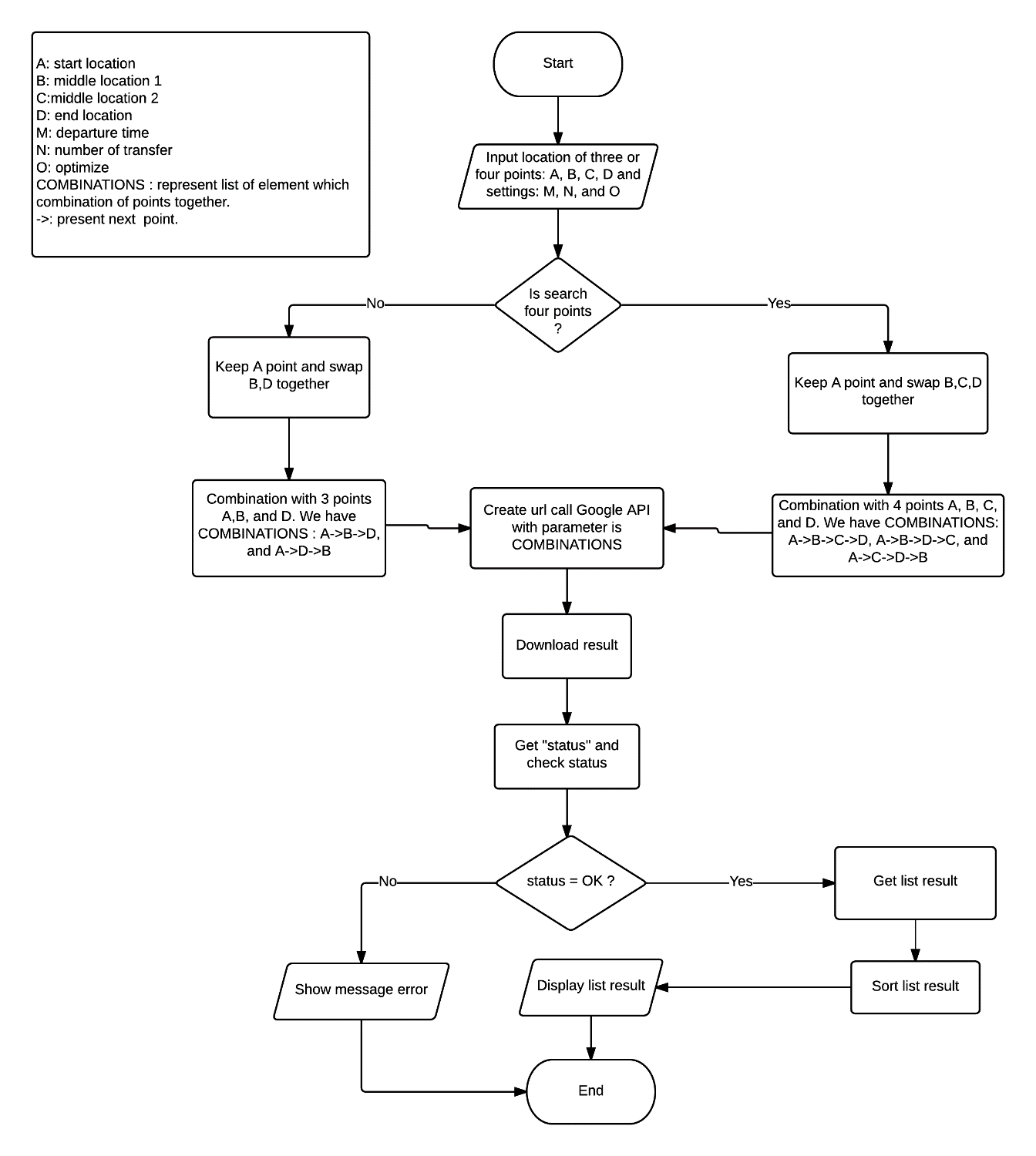
With four inputted point from user is A, B, C, D so we have the COMBINATIONS is ABCD, ABDC, ACDB, ACBD, ADCB, and ADBC. But we don’t care position of two middle locations so we may be reduced ABCD, ABDC, and ACDB.

* + Step 2: We use “Google API” to solve for each element in COMBINATIONS. When the Google Maps Directions API returns results, it places them within a (JSON) routes array. Even if the service returns no results (such as if the origin and/or destination doesn't exist) it still returns an empty routes array.
  + Step 3: Check status of result
    - If status = OK, get result from Google API
    - If status != OK, show message error.
  + Step 3: Parse result are returned from Google Map Direction API, we have list Leg.
  + Step 4: The list result is sorted follow two priorities below:
    - Time is high priority. We sort the list base on increasing time.
    - Distance is high priority. We sort the list base on increasing distance.

#### Algorithm complexity

* O(K): K is total number of requests had send to Google API.
* Three point: K = 2.
* Four point: K = 3.

#### Flowchart



### Distance from one point to segment algorithm

#### Problem definition

* How to calculate of user’s location to one segment on route?

#### Attribute definition

* Current location of user A: get location from Mobile’s GPS.
* Segment BC: that segment, which uses to calculate distance to current location. And it has start location B and end location C.
* Euclid distance of A to B is:
* Perpendicular:

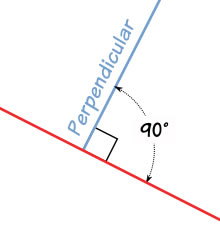


Figure 83: Perpendicular example

#### Solution

* Step 1: Get current location A and segment BC.
* Step 2: Map latitude and longitude of all location to x, y on 2D place (longitude -> x, latitude -> y) using Mercator projection.

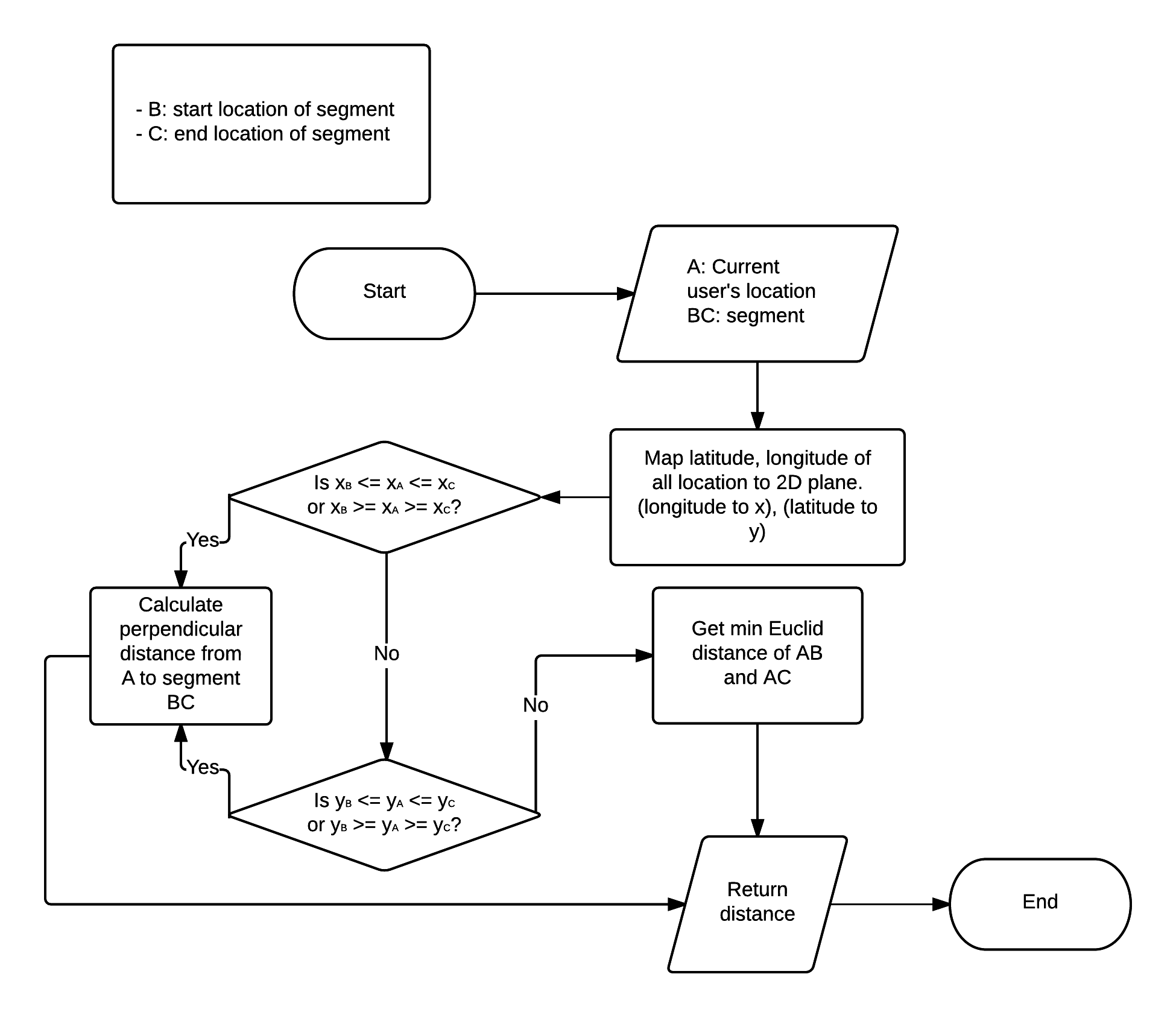
Reference: <https://en.wikipedia.org/wiki/Mercator_projection>

* Step 3: Is or or or ?
  + Yes: Calculate perpendicular distance from A to segment BC.
  + No: Get min Euclid distance of AB and AC.
* Step 4: Return distance.

#### Algorithm complexity

* O(1).

#### Flow chart



### Notify algorithm when on right route.

#### Problem definition

* Algorithm detects when users are on right route and sends notified signal for user preparing next turn.

#### Attribute definition

* IsTrueWay: variable for checking user currently is on true way or not.
* Current location of user A: get location from Mobile’s GPS.
* Current road: this road use to check current location.
* Allowed range: If distance from location of users to road is in this range, they will be on right route.
* Notified range: If distance from location of users to notified location is in this range, they will be notified.
* Euclid distance of A to B is:

#### Solution

* Step 1:
  + Get user’s location (A) from Mobile’s GPS.
  + Get current road.
  + Get allowed range.
  + Get notified range.
  + Get isTrueWay.
* Step 2: Check isTrueWay
  + True: go to next step.
  + False: Return isTrueWay and current road = null. End
* Step 3: Calculate all distances from user’s location to all segments of current road.
* Step 4: User will be in range if have one distance shorter than or equal allowed range. And opposite is out of range.
  + In range:
    - Map latitude and longitude of all location to x, y on 2D place (longitude -> x, latitude -> y) using Mercator projection.

Reference:

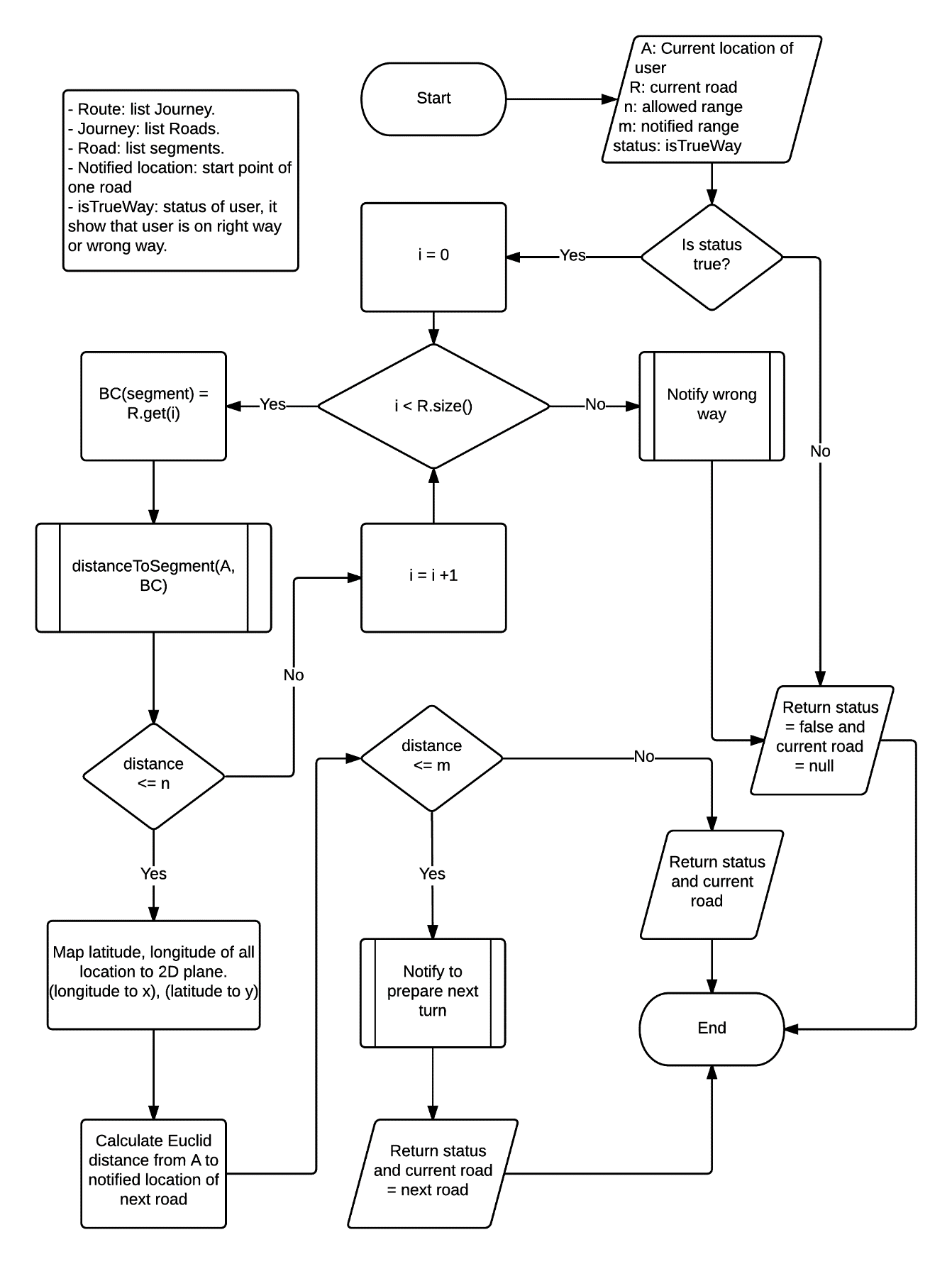
<https://en.wikipedia.org/wiki/Mercator_projection>

* + - Calculate Euclid distance from user’s location to notified location. Is distance shorter than or equal notified range.
      * Yes: system will notify to user. And he/she will know what should do next. Return isTrueWay and current road = next road. End
      * No: Return isTrueWay and current road. End
  + Out of range:
    - Notify to user that he/she is on wrong way.
    - Return isTrueWay = false and current road = null. End

#### Algorithm complexity

* O(N): N is total segments on road.

#### Flow chart



### Notify algorithm when on wrong route

#### Problem definition

* When user is far from searched route, system will change algorithm to wrong way state. In this state, algorithm will continue to detect wrong way and notify for user.

#### Attribute definition

* IsTrueWay: variable for checking user currently is on true way or not.
* Current location of user A: get location from Mobile’s GPS.
* ListAllSegments: list of all segments have in whole route.
* Allowed range: If distance from location of users to road is in this range, they will be on right route.
* Notified range: If distance from location of users to notified location is in this range, they will be notified.
* Euclid distance of A to B is:

#### Solution

* Step 1:
  + Get user’s location (A) from Mobile’s GPS.
  + Get listAllSegment.
  + Get allowed range.
  + Get notified range.
  + Get isTrueWay.
* Step 2: Check isTrueWay
  + True: go to step 3.
  + False: Return isTrueWay and current road = null. End
* Step 3: Calculate all distances from user’s location to all segments of listAllSegment. Go to step 4.
* Step 4: User will be in range if have one distance shorter than or equal allowed range. And opposite is out of range.
  + In range: Call segment, which has shortest distance to user’s location is segment (BC). Change variable of isTrueWay to true. Go to step 5.
  + Out of range: Return isTrueWay and current road = null. End.
* Step 5: Map latitude and longitude of all location to x, y on 2D place (longitude -> x, latitude -> y) using Mercator projection. Go to step 6.

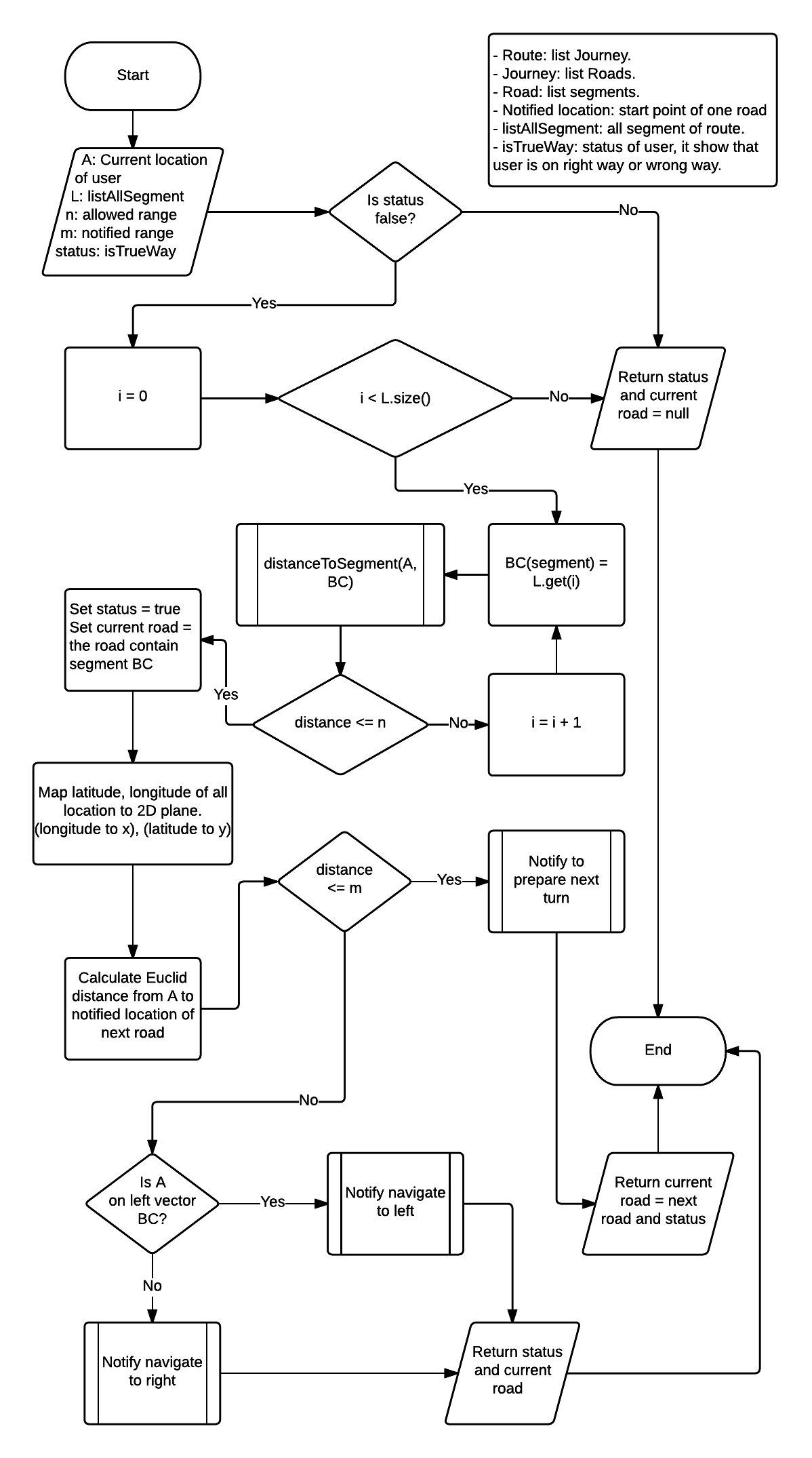
Reference: <https://en.wikipedia.org/wiki/Mercator_projection>

* Step 6: Calculate Euclid distance from user’s location to notified location.
  + If distance is shorter than equal notified range:
    - System will notify to user. And he/she will know what should do next.
    - Return isTrueWay and current road. End
  + If distance is longer than notified range: Go to step 7.
* Step 7: Map latitude and longitude of A, B and C to x, y on 2D place (longitude -> x, latitude -> y) using Mercator projection. Go to step 8.
* Step 8: Check position of A to vector is left or right.
  + Right: Notify to user that they should navigate to right direction. Go to step 9.
  + Left: Notify to user that they should navigate to left direction. Go to step 9.
* Step 9: Return isTrueWay and current road. End

#### Algorithm complexity

* O(N): N is total segments on whole route.

#### Flowchart



### Download audio file algorithm

#### Problem definition

* When participating traffic, user needs to notify with voice. So, we have to convert message to audio file and save audio file to disk cache of mobile.

#### Attribute definition

* FPT Service: convert from message string to audio file in the form (array byte).
* DiskLruCache: It is library save image into cache provided by Jake Wharton, please reference to link <https://github.com/JakeWharton/DiskLruCache> to view detail. With mechanism a cache that uses a bounded amount of space on a file system. Each cache entry has a string key and a fixed number of values. Each key must match the regex [a-z0-9\_-] {1, 64}. Values are byte sequences, accessible as streams or files. Each value must be between 0 and Integer. MAX\_VALUE bytes in length.

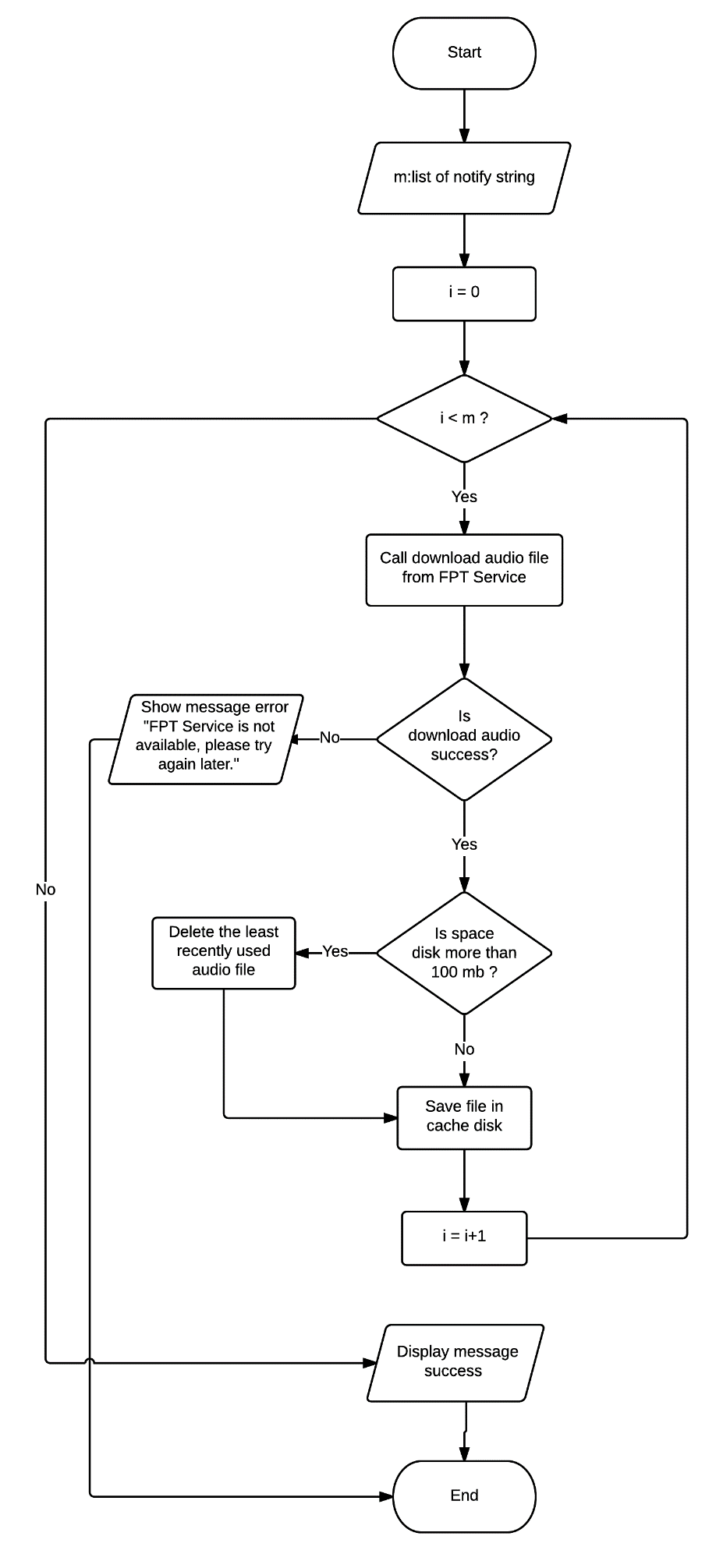
#### Solution

* FPT Service return byte [] data, while DiskLruCache save image in the format is bitmap. So, we need edit DiskLruCache library to save in the format is byte [].
* To solve the problem downloading audio file, we follow these steps:
* Step 1: We have list string notify from search route.
* Step2: Call FPT Service with Url [http://118.69.135.22/synthesis/file?voiceType=female&text= + message](http://118.69.135.22/synthesis/file?voiceType=female&text=%20+%20message) to download audio file and save into disk cache of mobile.
* If download fail, system show message notify “FPT Service is not available, please try again later.”
* If download success, system will check space disk.
* Step 3: Check space disk.
  + If space disk is greater than 100 mb then check audio file least recent use delete it and save new audio file into cache disk.
  + If space disk is not greater than 100 mb then save new audio file into cache disk.
* Step 4: Show message download success when process is finish.

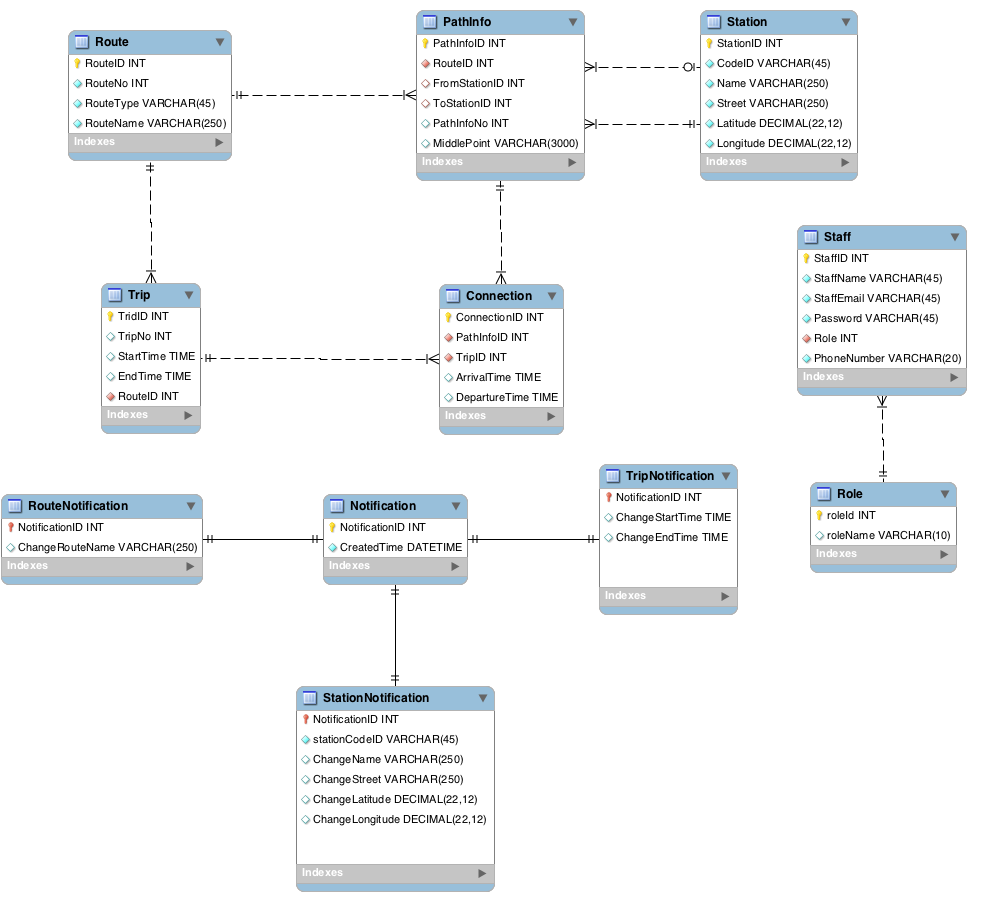
#### Algorithm complexity

* O(N): N is total audio files need to download.

#### Flow chart



## 2. Database Relationship Diagram



# Appendix

1, RAPTOR algorithm is based on paper “Round-Based Public Transit Routing” written by Daniel Delling, Renato F. Werneck (Microsoft Research Silicon Valley), Thomas Pajor (Karlsruhe Institute of Technology), public in 2012.

1. [↑](#endnote-ref-1)