SE2XB3 Requirements Specification: OptimizeU

SE2XB3 Group 8, L03

February 25, 2018

Contents

1 Introduction

The purpose of this document is to provide a requirements specification for building an optimization system for Uber drivers to increase maximum uptime for drivers and passengers alike.

2 Domain

2.1 Application

OptimizeU will have the following information:

- **Uber pickups Dataset**:
  Used to generate cluster information and pickup hotspots for drivers to locate and pickup passengers
- **Client-Server Interface**:
  Information is stored server-side. Users (drivers) fetch data from the server by input from the client.

A more detailed functional and non-functional specification will be discussed later below.

2.2 Stakeholders

The stakeholders and their main relationships are listed below:

- **Drivers**:
  These are the primary users of the application; They interact directly with the program by obtaining location hotspots from certain times of days to get optimal travel routes and cluster visitations.
- **Passengers**:
  Although passengers do not directly access the program, they are also considered a major stakeholder due to their relationship with drivers; passengers are the customers for the services that drivers provide.

2.3 Goals, Expectations and Effects

- **Maximize driver's uptime and profits**:
  By smart optimization of driver's recommended locations and times to visit, more time will be spent on fetching passengers instead of finding for potential customers.

- **Improve passenger's wait times and safety**:
  Since more drivers are directed to passenger hotspots by usage of OptimizeU, passengers can enjoy minimized wait times and also street crime can be indirectly reduced as passengers spend less time waiting outside on streets.

3 Functional Requirements

The functional requirements will be documented using a use-case document (in the form of a table), and a use-case diagram.

Name:	Get Optimized Routes
Created By:	SE2XB3 2018 Group 8
Description:	Uber driver submits a request for optimized routes to OptimizeU Server
Actors:	Uber driver, OptimizeU Server, OptimizeU Client User Interface
Preconditions:	1. Uber driver has access to the Client User Interface 2. OptimizeU has fully functional Client and Server 3. Driver is in New York City area 4. Dataset and server security is not compromised
Postconditions:	1. Client retrieves and displays requested information successfully 2. Drivers get optimized routes from OptimizeU
Flow:	1. Driver makes a request to OptimizeU server through the client 2. OptimizeU sends information about driver's location and timeframe to server 3. Server fetches dataset 4. Server uses dataset and driver information to process an optimized output 5. Server sends output information back to client 6. Driver uses information received as their driving route
Alternate Flow:	In Step 2, if driver's time is abnormal (e.g. past midnight times) 1. Proceed as normal up to step 5 2. Send a warning about possibly inaccurate information together with the output data back to the client 3. Display warning to driver

Table 1: Use Case Document in Table form

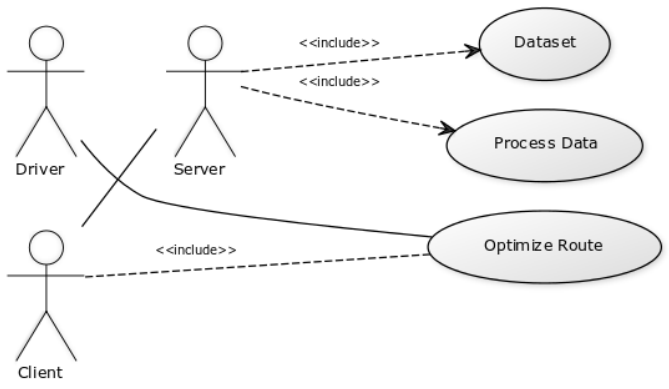

Figure 1: Uses Case Diagram

4 Non-Functional Requirements

4.1 Performance and Accuracy of Results

OptimizeU should have the following capabilities:

- **Adequate server-client response time:**
  Users should expect to get results within a reasonable time frame after submitting a request to the OptimizeU server. Depending on server load and external factors, users should expect a response within 10 to 15 seconds.

- **Consistent, accurate output**
  OptimizeU should consistently generate meaningful output for drivers to act on. An example being clustering that makes sense (e.g. a mega-cluster covering an entire city wouldn't be of any use); or paths that are always minimal.

4.2 Software Quality Attributes

OptimizeU should possess the following software quality attributes:

- **Reliability:**
  Generate optimized paths that are actually of use to the user. See the above section on consistency for an example.

- **Security:**
  Since this is a client-server interface, users should be assured that any sensitive

information sent (for example, location) is properly handled and safeguarded against malicious intent. This can probably be done via HTTPS. The server should also be protected against attacks, but this will not be the main focus of the project.

- **Safety:**
  OptimizeU should ensure driver safety by providing appropriate route information. For example, routes should obey flow-of-traffic rules. This will probably be implemented as an extra feature, as this is not in scope of the project.

- **Portability Issues:**
  OptimizeU is portable client-side as it is implemented using HTML. However, programmers may have to account for server-side context, but it should not be an issue since it is implemented using portable Java code.

5 Requirements on the Development and Maintenance Process

5.1 System test procedures

The following system test must be carried out before the final product is ready:

- **Algorithm testing**
  Test critical algorithms rigorously to ensure that accurate information is always delivered to users.

- **Server testing**
  Test that the server functions as intended to process client input and sends correct information.

- **Client interface testing**
  Ensure that client interface are standardized to HTML and provide an acceptable user experience.

5.2 Priorities of Required Functions

A list of required functions (on server backend) and their corresponding priorities are listed:

- **k-medoids**
  The clustering algorithm that is planned to be implemented. Must work on every occasion save on faulty or abnormal user input

- **Kruskal**
  The minimum-spanning tree path generation. Dependent on the correctness of k-medoids, but must also properly generate a MST on every clusters.

5.3 Likely changes to System Maintenance Procedures

- Seperate tests may be carried out on similar algorithms (e.g. Prim vs Kruskal, k-means vs k-medoids etc.) if performance suffers. Change if necessary as well.
- Client-side maintenance as an iterative process: Since user experience relies on feedback to a certain extent, the client-side interface may be constantly modified, On each iteration, its communication with the server must always be tested to ensure flawless communication.
- Security testing: Testing to check server compromisation should be carried out AND MODIFIED periodically to ensure that attackers do not gain improper access to server-side information or user information.