

GROUP SW101F14

MASTER'S THESIS

---

# GPS-Based Road Pricing

---

*Authors:*

Mikael MIDTGAARD

Jens MOHR MORTENSEN

Dan STENHOLT MØLLER

*Supervisor:*

Kristian TORP

June 4, 2014



## **Abstract**

GPS-Based Road Pricing is a system comprised of all components required of a complete fully-functional GPS-based road pricing system. The system consists of a mobile client, web client, server, and databases. The mobile client acts as an onboard unit that is able to gather GPS locations of a driver. These locations are used for calculating the road pricing tax of the driver. The accumulated tax is calculated online, which allows it to be displayed to the driver in real-time. The web client provides extensive information of the road pricing tax for the drivers of the system.

The system is developed in such a way that it can support millions of drivers, which makes it possible to perform large scale experiments. Field trials are performed to verify that the system is fully-functional, and satisfies all requirements of a complete GPS-based road pricing system.





**AALBORG UNIVERSITY**  
STUDENT REPORT

**Title:** GPS-Based Road Pricing  
**Subject:** Database Technology  
**Semester:** Spring 2014  
**Project group:** sw101f14

**Authors:**

---

Mikael Midtgaard

---

Jens Mohr Mortensen

---

Dan Stenholt Møller

**Supervisor:**

Kristian Torp

**Number of copies:** 5

**Number of pages:** 89

**Number of appendices:** 5

**Completed:** June 4, 2014

**Department of Computer Science**  
**Aalborg University**  
Selma Lagerlöfs Vej 300  
DK-9220 Aalborg Øst  
Telephone +45 9940 9940  
Telefax +45 9940 9798  
<http://cs.aau.dk>

**Synopsis:**

GPS-Based Road Pricing is a system that consists of all components required for running and experimenting with large scale real-world GPS-based road pricing. The system consists of an Android app, a web application, a server, and four databases. The Android app performs online road pricing calculations and displays the cost to the driver and reports it to the server. The web application displays specific information on the road pricing costs. The server handles requests from the Android app and web application, and stores and retrieves data from the appropriate databases.

Field trials are performed to showcase and illustrate the accuracy of the complete functioning road pricing system. Privacy of the users is handled by encrypting all data traffic and storing privacy-sensitive information separately. The costs of operating the road pricing system is minimized by using a novel message format. The system is able to scale to millions of users because of the low data costs and because calculations are performed on the mobile client.

*The content of this report is freely accessible. Publication (with source reference) can only happen with the acknowledgment from the authors of this report.*



# Preface

This Master's thesis is written by three students on the 4<sup>th</sup> and final semester of the Master of Science (MSc) in Engineering (Software) education at Aalborg University in the spring of 2014. The theme of the project is database technology. Recurring terms are defined in the glossary.

Whenever a technical choice is taken it is accompanied by a comparison table. These tables consist of the different technology options that are considered for each component. Each component is graded on a given set of aspects ranging from one (★) to three stars (★★★), where one is worst and three is best. Based on these requirements we will decide which choice is most suitable for our project.

In Appendix E a CD is attached that contains:

- The source code of the mobile app, web application, and web services.
- The database construction queries.
- The map and taxation model database dumps.
- README.txt file explaining how to use the GPS-Based Road Pricing system.

Alternatively, the contents of the CD can be accessed at  
<http://130.225.198.79:6060/RPServer/files/CD.rar>





# Glossary

Terms that are used throughout this project are defined in the list below.

- **Road Pricing System** – All the components necessary to perform GPS-based road pricing.
- **The System** – Same as Road Pricing System.
- **Road Pricing Provider** – The organization in charge of running and maintaining the road pricing system.
- **Taxation Model** – Encapsulates the rules of an entire road pricing system.
- **Billing** – Describes what a driver is paying in road pricing tax as a list of time intervals with a price associated with each interval.
- **Segment** – A part of a road network.
- **Travel Log** – A price associated with a location represented as as segment at a specific time
- **Trip** – A path traveled by a driver, described as a list of segments.
- **Map matching** – Associating a GPS location to a single segment in order to determine where the driver is.
- **Onboard unit** – A device installed in a vehicle able to acquire and process data.



# Contents

	Page
<b>I Introduction</b>	<b>1</b>
<b>1 Motivation</b>	<b>3</b>
<b>2 Problem Definition</b>	<b>5</b>
2.1 Problem Statement . . . . .	5
2.2 Problem Limitation . . . . .	5
<b>3 Analysis</b>	<b>9</b>
3.1 Privacy . . . . .	9
3.2 Client/Server Functionality . . . . .	10
<b>4 System Definition</b>	<b>13</b>
<b>5 Technologies</b>	<b>15</b>
5.1 Mobile Client . . . . .	15
5.2 Server . . . . .	16
5.3 Web Client . . . . .	21
5.4 Databases . . . . .	21
<b>6 Data Usage</b>	<b>25</b>
6.1 Send Billing information . . . . .	25
6.2 Send Travel Information . . . . .	26
<b>II Design</b>	<b>27</b>
<b>7 System Architecture</b>	<b>29</b>
7.1 Mobile Client . . . . .	30

7.2	Server . . . . .	31
7.3	Web Client . . . . .	32
7.4	Taxation Model Database . . . . .	32
7.5	Billing Database . . . . .	33
7.6	Encrypted Travel Log Database . . . . .	34
7.7	User Database . . . . .	34
<b>8</b>	<b>Client GUIs</b>	<b>37</b>
8.1	Mobile App . . . . .	37
8.2	Web Application . . . . .	41
<b>9</b>	<b>Database Structures</b>	<b>43</b>
9.1	Taxation Model Database . . . . .	43
9.2	Billing Database . . . . .	43
9.3	Encrypted Travel Log Database . . . . .	44
9.4	User Database . . . . .	45
9.5	Local Database . . . . .	46
<b>III</b>	<b>Implementation</b>	<b>47</b>
<b>10</b>	<b>Mobile App</b>	<b>49</b>
10.1	Activities . . . . .	49
10.2	Services . . . . .	51
10.3	Databases . . . . .	55
<b>11</b>	<b>Web Application</b>	<b>59</b>
11.1	Login . . . . .	59
11.2	Travel Log . . . . .	60
11.3	Billing . . . . .	61
11.4	Map . . . . .	61
11.5	Filtering . . . . .	62
<b>12</b>	<b>Web Services</b>	<b>63</b>
12.1	getApplicationInfo . . . . .	63
12.2	sendBilling . . . . .	64
12.3	sendTravelLog . . . . .	65
<b>IV</b>	<b>Evaluation</b>	<b>67</b>
<b>13</b>	<b>Results</b>	<b>69</b>
13.1	Field Trial . . . . .	69
13.2	Message Size . . . . .	71
13.3	Data Usage . . . . .	74

<b>14 Discussion</b>	<b>77</b>
14.1 Field Trial Experiences . . . . .	77
14.2 Accuracy in Map Matching . . . . .	78
14.3 Scalability . . . . .	79
14.4 Setup and Operational Costs . . . . .	79
<b>15 Conclusion</b>	<b>81</b>
<b>16 Future Work</b>	<b>83</b>
16.1 Map Matching . . . . .	83
16.2 Data Encryption . . . . .	83
16.3 User Management . . . . .	84
16.4 Message Reduction . . . . .	84
16.5 Fraud Prevention . . . . .	84
<b>Bibliography</b>	<b>85</b>
<b>V Appendices</b>	<b>91</b>
<b>A Taxation Model Database Schema</b>	<b>93</b>
<b>B Travel Data</b>	<b>95</b>
<b>C Data Usage</b>	<b>97</b>
<b>D Data Usage Calculations</b>	<b>99</b>
<b>E CD</b>	<b>101</b>

**Part I**

**Introduction**



## Motivation

Several types of road pricing are in effect today [41, p. 10]. These include vignette tax [53][19], congestion zones [8], and toll roads such as the Great Belt Fixed Link [52] and the Oresund Bridge [36].

Experiments have been performed with more sophisticated methods of road pricing such as GPS-based road pricing, and in 2005 the first GPS-based road pricing system was launched: LKW-Maut [7]. LKW-Maut is a toll for trucks in Germany with a maximum weight of 12 metric tons and above [49]. An onboard unit is used to measure the distance traveled.

Experiments have been conducted to implement GPS-based road pricing for cars in the United Kingdom, Holland, Singapore, and Denmark [54, pp. 128-129]. In the United Kingdom the “Road Pricing Demonstrations Project” was conducted from 2008 to 2011 [16]. In this demonstration project four companies – Intelligent Mechatronic Systems, Sanef Tolling Limited, T-Systems Ltd, and Trafficmaster plc – each tested road pricing on 100 volunteer drivers. An additional three companies – Kapsch TrafficCom Limited, Q-Free ASA, and Serco Ltd – assisted in developing road pricing systems. Four different onboard units were used in the “Road Pricing Demonstrations Project”: three thick clients and one thin client [17]. A thin client only gathers time and location data, and then relays this to another unit that calculates the road pricing cost. A thick client gathers the same data as a thin client, but also calculates the road pricing cost.

These four devices were specifically designed for GPS-based road pricing. Therefore, the number of test drivers is restricted to the number of prototype devices the companies have produced. No one has created a system for a publicly available platform that allows for large scale experiments. We intend to create such a system.





# Problem Definition

In this chapter we present and analyze the problem at hand.

## 2.1 Problem Statement

Based on the previous chapter we define our problem statement as:

*How can we analyze, design, and implement all components required for a fully functioning GPS-based road pricing system, such that we are able to make large scale real-world experiments with a complete road pricing system?*

## 2.2 Problem Limitation

In this section we expand upon our problem statement by setting goals and assumptions for the road pricing system.

### 2.2.1 Goals

We define five goals for the road pricing system we develop. Being able to showcase the system; high accuracy; high privacy; low startup and operational costs; and high scalability. In the following subsections we describe the meaning of each goal.

#### Showcasing

An estimated cost of driving should be accessible for drivers during a trip to allow for an immediate understanding of how road pricing will affect them.

Drivers should be able to see a complete list of prices associated with their temporal and spatial travel data, so they can see exactly what they are paying for.

### **High Accuracy**

It is important to know where drivers are, such that they pay the correct road pricing tax. The GPS receiver in the onboard unit should be able to report locations so accurately that it is possible to correctly map match the GPS location to the segment the driver is located on.

Additionally, if a driver is not connected to the Internet, or if communication with the server temporarily fails no information should be lost.

### **High Privacy**

The system handles spatial and temporal data of the travel patterns of drivers. This means that drivers supply information on where their vehicles are at all times when driving. We should ensure that such data remains private.

Data transfers between the driver and the server should be encrypted, such that third parties cannot read the data.

The data should be stored such that the spatial data of the driver is inaccessible to the road pricing provider. The road pricing provider should only have access to how much a driver should pay in road pricing tax.

### **Low Startup and operational costs**

In order to compete with other types of road pricing, the road pricing system we propose should have a low monetary cost. The startup cost of road pricing should be low compared to other types of road pricing. The operational cost for the road pricing system should be a small fraction of the revenue generated by road pricing.

### **High Scalability**

The road pricing system should be able to handle many users. This means that the pressure on the network should be minimized. The road pricing system should be easily expandable to allow for more users.

## **2.2.2 Assumptions**

We make some assumptions concerning the road pricing system. They concern issues that are out of our control or would be too time-demanding for us to handle.

### **Honest users**

We assume that the users of the system are not attempting to cheat. This means that users do not tamper with the device installed in vehicles, attempt to report false locations, or interfere with the running software.

### Correct map data

We assume that the map data covers all segments covered by the taxation model. We assume that the map data is up-to-date with changes to the road network. We assume that the map data that we are using is fully segmented, which means a new segment begins at each intersection. Using fully segmented map data means that when a driver enters a new segment, he will drive the entire length of the segment unless he ends or begins his trip at that segment. This concept is illustrated in Figure 2.2.1, where each segment is illustrated with a distinct color. When driving along the blue segment from left to right in road network (a), the driver encounters a new segment at the intersection. This is not the case in road network (b). This means that road network (a) is fully segmented and road network (b) is not.

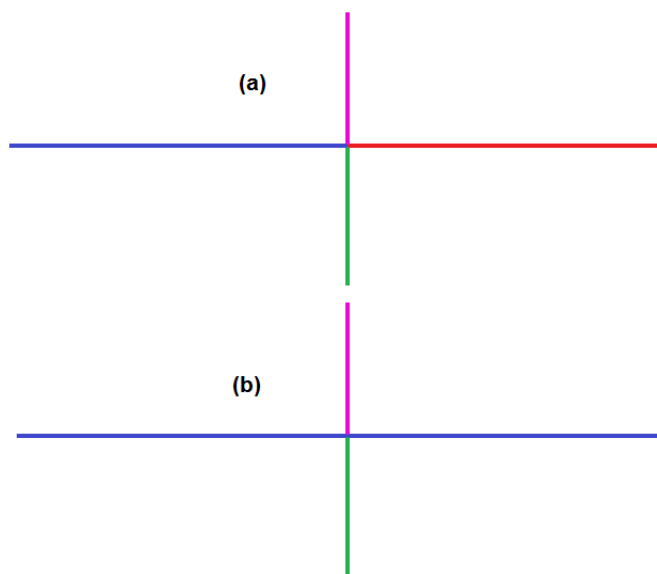


Figure 2.2.1: A fully segmented road network (a), and a road network that is not fully segmented (b). A different color is used for each segment.



# Analysis

In this chapter we analyze the possible choices for the road pricing system we develop. We base our choices on how well they reflect the goals described in Section 2.2.1.

## 3.1 Privacy

In this section the privacy issues of the system will be analyzed. We analyze these issues because of the privacy goal from Section 2.2.1. We analyze two privacy issues: data transfer and data storage.

### 3.1.1 Data Transfer

In the road pricing system, information will be transferred from the server to the client and vice versa. Some of the information is subject to privacy concerns, e.g. most people are not interested in exposing when they visit the doctor. As described in our privacy goal in Section 2.2.1 we want to ensure that such data remains private. It is undesirable that third parties can read the data while it is being transferred. Analysis on more data would be able to yield the travel patterns of a driver. To avoid this we encrypt the data that is transferred between the client and the server.

### 3.1.2 Data Storage

The data that the server receives is stored in a database. Since the location data of the drivers is sensitive this needs to be encrypted in the database. However, not all the stored data is equally sensitive. We do not consider how much road pricing tax a driver pays to be as sensitive as the location data of a driver. The road pricing provider should be able to see the billing information for a driver, but should not be able to see the locations the driver. The billing information and travel information should be stored in different databases to differentiate between them. Furthermore, the travel information should be stored encrypted.

## 3.2 Client/Server Functionality

In this section we discuss the distribution of functionality between the client and the server. In the road pricing system the onboard unit is the client. We discuss the distribution because it has a direct effect on the scalability, privacy, and cost goals from Section 2.2.1. The following topics are important when it comes to functionality distribution: Price calculation, taxation model, map, and history.

The most important topic is to decide if the price calculation should be performed on the client or on the server, because the calculation is based on a taxation model and a map. This means that the taxation model and map have to be stored where the price calculation is performed.

The effects of having price calculation on the client and the server will now be outlined.

### 3.2.1 Client-side Calculation

In order to perform the price calculations client-side there has to be a local database on each client. This database should contain map data and a taxation model that describes the cost of driving. In order to determine the road pricing tax of the driver, the client has to establish where the driver is located based on GPS locations – this is known as map matching. When map matching is performed on the client the cost of sending GPS-locations to the server and receiving road segment IDs is eliminated; this has the added effect of increased privacy since the exact locations of the driver are not sent. Furthermore, response time is faster, since time spent on data transfer is eliminated. Similarly the traffic and response time costs are reduced when price calculations are performed based on a taxation model that is stored on the client.

When the price calculations are performed on the client it alleviates the processing requirements on the server-side.

### 3.2.2 Server-side Calculation

Calculating the cost of driving requires spatial and temporal data of the travel patterns of the driver. This means that the client has to either send GPS locations or road segment IDs to the server depending on where map matching occurs. If map matching is done on the client, segment IDs are sent. If map matching is done on the server GPS locations are sent.

There are significantly more GPS locations than segment IDs, since more GPS locations are map matched to the same road segment. This means that the data traffic cost is higher if map matching is done on the server. Additionally, the privacy will be more difficult to handle since the exact locations of the driver are sent to the server.

If map matching is done on the server, the taxation model and map data have to be stored on the server. This means that the taxation model and map data do not have to be stored on the client. This would reduce the overall space requirement tremendously, since the taxation model and map data would only have to be stored in a single location as opposed to potentially millions.

If there are many users, the server requirements for calculating the road pricing tax is high. This means that calculating the road pricing tax on the server scales poorly compared to the client-side approach.

### 3.2.3 Choosing Price Calculation Location

Choosing the location of the price calculation should reflect our goal for a scalable, private, and low cost system, as described in Section 2.2.1. The following factors are important to such a system: overall space requirements, data traffic, response time, privacy, and scalability.

	Client-side	Server-side
Overall space requirements	★	★★★★
Data traffic	★★★★	★
Response time	★★★★	★
Privacy	★★★★	★
Scalability	★★★★	★
Total	13	7

Table 3.2.1: Comparison table between client-side and server-side price calculation.

The space requirements are the same on a client and on a server. The difference is that if the map data and taxation model are stored client-side, it has to be stored on all clients instead of only once on the server. Since there are potentially millions of clients the overall space requirements are significantly higher when the price calculation is performed client-side.

If the price calculation is performed on the server the data traffic cost is much higher, and privacy becomes more difficult to handle.

If the price calculation is performed on the client it has to communicate with the server fewer times, which means that the response time of the application is higher.

We choose to perform price calculations on the client because the data traffic cost will be lower; the response time will be lower; privacy will be easier to handle; and the system scales better.





## System Definition

In this chapter we define the system we develop. We define the system based on the problem statement from Section 2.1, the goals we defined in Section 2.2.1, and the analysis made in Chapter 3.

The system implements road pricing.

It does so by running a mobile app on a mobile client installed in road vehicles that transfer data to a server. On the mobile client a taxation model and map data are stored according to the region of the driver. This means that the system can be used anywhere in the world, as long as the correct taxation model and map data are stored on the client.

The mobile client generates data based on the location of the driver. The data is processed and stored on the mobile client before it is sent to the server. There are two types of data that is sent to the server: travel log data and billing data. The travel log data describes where a driver was at a specific time and the cost thereof. The billing data describes what the driver pays in road pricing tax in a specific time interval. Before the data is transferred from the mobile client to the server it is stored on the mobile client. The two types of data are sent to the server in distinct intervals.

The data traffic is encrypted to avoid third parties from accessing it. The server stores travel log data and billing data separately.

It is possible for the drivers to access their travel log information and billing data on a web client. It is possible for the road pricing providers to access billing information for any driver in the system on a web client.



# Technologies

In this chapter we analyze the available technologies for the components of the system. Based on the system definition in Chapter 4 we need to analyze technologies for the following components: a mobile client, web client, server, and databases.

The available technologies and the choice of the most suited for each component will be outlined in the following sections.

## 5.1 Mobile Client

In this section we present an analysis of which mobile client the mobile app is most suitable to run on.

The primary use of the mobile client is to track the position of the drivers as they travel in order to calculate the road pricing tax.

The mobile app needs to run on a mobile client for which we have certain requirements: It needs to have a GPS receiver in order to track the location of the drivers. In order to keep traffic costs low the mobile client needs to perform some calculations, such that it is not dependent on receiving results from a server, as described in Section 3.2. The client will, however, still need to communicate with a server. The mobile client needs to communicate with a server from nearly any location, since it is installed in a road vehicle.

The mobile client needs a screen to showcase the cost to the driver. Since the mobile client needs to be installed in all road vehicles, it needs to be low in price.

Finally, we must be able to develop an app that runs on the client.

We consider two types of clients: smartphone and dedicated device.

Modern smartphones satisfy all these requirements: They have GPS receivers, relatively powerful processors, are cheap, easily accessible, are able to communicate with a server using mobile networks such as 3G, and have screens.

In Table 5.1.1 a comparison between smartphones and dedicated devices can be seen. We choose a smartphone as the onboard unit as it scores the highest in the comparison.

	<b>Smartphone</b>	<b>Dedicated Device</b>
GPS	★★★	★★★
Communication to server	★★★	★★★
Programmable	★★★	★
Screen	★★★	★
Acquisition and installation price	★★★	★★★
<b>Total</b>	<b>15</b>	<b>11</b>

Table 5.1.1: Comparison between mobile client.

Another advantage of using a smartphone as the mobile client is that there are development kits available.

For Android, which is the most popular operating system for smartphones [4][5], Android SDK is freely available [27].

For iOS, which is the second most popular operating system for smartphones [4], Xcode is freely available [24]. However Xcode is only compatible with OS X.

We have considerable experience with Android development, and very limited experience with iOS development. We have immediate access to Android smartphones, which is not the case for iPhones.

	<b>Android</b>	<b>iOS</b>
Market share	★★★	★★
Development kit	★★★	★★★
Experience	★★★	★
Accessibility	★★★	★
<b>Total</b>	<b>12</b>	<b>7</b>

Table 5.1.2: Comparison between smartphone operating systems.

As seen in Table 5.1.2 Android scores the highest and therefore we choose this as the operating system for the mobile client. We choose to develop an Android 4.0 app because in this version Google combined the tablet and phone Android versions into one [26]. This means that some core functionality is performed differently from older Android versions.

By choosing 4.0 we allow for more people to use our mobile app, than if we had chosen a newer version.

## 5.2 Server

The server is the connection between the databases and the clients in our system. This connection will be provided in the form of web services, which is a method for communicating between two clients over the Internet.

Since we choose to develop the mobile client as an Android app it would be natural to select Java as technology for the application server that runs on the server. The reason for this is that Android apps are developed in Java and we can thereby reuse the models from the Android app on the application server.

In the following sections the technology choice for the application server and the web service interface will be analyzed.

### 5.2.1 Java Application Server

Several Java application servers exist, but we will only consider the following.

#### **Jetty** [13]

Is an application server with a relatively small download size of approximately 10MB. It has plugins to most well-known IDEs. Configuration of the server is done by sending XML based configuration files along when starting the server. The team behind Jetty has created Jetty Documentation Hub, which is used to provide support for their server. Jetty is not rich on server features. It is basically just a container. The Jetty application server is free to use under the Apache 2.0 license.

#### **GlashFish** [43]

Is an application server that has a download size of approximately 50MB. It has plugins to most well-known IDEs. Configuration of the server can be done in one big XML file. Glassfish has documentation available on their website. Glassfish includes all the server features of a full Java EE compliant application server. The Glassfish application server is free to use, but can be extended with a license that includes additional features such as 24/7 support.

#### **Tomcat** [6]

Is an application server, which is lightweight, and has a download size of approximately 12MB. It has plugins to most well-known IDEs. Configuration of the server is done in several XML files, but mostly in the server.xml file. Tomcat has a big community and it is one of the most used Java application servers, so it is easy to find support on a problem [35]. Tomcat is low on server features. The Tomcat application server is free to use under the Apache 2.0 license.

#### **JBoss** [23]

Is the application server that has the biggest download size of approximately 130MB. It has plugins to most well-known IDEs. JBoss has a big community, which is a big help when in need of support. Configuration of the server is done in one big XML file called standalone.xml. Jboss includes all the server features of a full Java EE compliant application server. The JBoss application server is free to use, but there is an enterprise version that includes additional support.

### 5.2.1.1 Technology Choice

In Table 5.2.1 the comparison between the application servers can be seen. The performance field is based on the test made by Maple [38].

	Jetty	GlassFish	Tomcat	JBoss
Installation Size	★★★	★★	★★★	★
Server Configuration	★★	★★★	★★★	★★★
Documentation and Community	★★	★★	★★★	★★
Features	★	★★★	★	★★★
Performance	★	★	★★★	★★
Total	9	11	13	11

Table 5.2.1: Comparison between the Java application servers.

We choose to use Tomcat as our application server, because it is dominant in all fields except the feature field. This is not a noticeable lack, since we will not be using all the features a full Java EE compliant application server offers.

### 5.2.2 Web Service Interface

For the creation of a web service there are two interfaces that are commonly used; these are SOAP (Simple Object Access Protocol) [9] and REST (REpresentational State Transfer) [14].

SOAP is a message protocol that is based on XML [10], the protocol defines a message architecture and format. SOAP defines a top level XML element called an envelope which itself consists of two elements: a header and a body. The header of the SOAP envelope is extensible and can contain information that can be used for routing purposes. The body of the SOAP envelope contains the message that are to be transmitted between the two clients.

One of the advantages of SOAP is that it uses the well defined XML message format, which means that SOAP messages always look the same. This on the other hand also makes the messages increase in size and therefore more bandwidth usage is required.

REST is a architectural style that is based upon HTTP [32]. In REST resources are identified with an URI that can be accessed trough the HTTP methods GET, PUT, POST, and DELETE. The messages in REST are decoupled from their representation. This means that the messages are not limited to XML.

One of the advantages of REST is that the developer can choose the format of the message. This means that if a lightweight message is needed, it can be accomplished by choosing a message format that uses few bytes.

One of the disadvantages of REST is that it is an architectural style which means that it only consists of best practices and there are no rules enforcing a best practice.

Pautasso et al. [46] gives an in depth look at SOAP and REST with an analysis of which interface to choose in different cases. REST is more lightweight because there is

no technology stack to setup, whereas SOAP depends on a large technology stack that needs to be setup.

### 5.2.2.1 Technology Choice

In Table 5.2.2 the comparison between the server interfaces can be seen. The latency field is based on the test made by Aihkivalo and Paaso [3, p. 408].

	SOAP	REST
Web Service Stack Size	★	★★★
Message Format Structure	★★★	★
Message Byte Size	★	★★★
Available Message Formats	★	★★★
Latency	★	★★★
Total	7	13

Table 5.2.2: Comparison between the web service interfaces.

We choose to use REST as our server interface, because with REST we can reduce the data traffic, which is in line with our scalability and cost goals, as described in Section 2.2.1. Because of this decision we have multiple message formats available, and therefore we need to decide on which format to use. The decision will be discussed in the following section.

### 5.2.3 Message Format

As mentioned in the previous section we have the option to use a wide variety of message formats for our REST implementation. We have decided to choose between XML, JSON [31], Protocol Buffers [20], and a novel message format.

It is possible to compare these message formats in a number of ways and we have decided to compare the following points: readability, marshalling speed, and payload size.

We have decided to include a novel message format in the comparison because this will give us more control over the data flow between the client and the server.

Aihkivalo and Paaso [2] look at different implementations of marshalling and unmarshalling. This is of interest as this is an action that is performed every time a message is transmitted. In this report they find that there is no solution that performs best in all cases.

Aihkivalo and Paaso [3] look at the difference in payload and transmission time for XML, JSON and Protocol Buffers. They conclude that XML has the biggest payload and protocol buffers the smallest. The payload for XML and JSON can be reduced using gzip [37].

Ismail [33] looks at the advantages and disadvantage of protocol buffers, which is an emerging choice of formatting. He concludes that even though protocol buffers has the



advantages of being faster marshalled and smaller in payload, it suffers because it is not readable and does not support inheritance or polymorphism. This drawback means that it is hard to debug and errors have to be parsed to be readable.

### 5.2.3.1 Technology Choice

In Table 5.2.3 the comparison between the message formats can be seen.

	<b>XML</b>	<b>JSON</b>	<b>Protocol Buffers</b>	<b>Novel Message Format</b>
Readability	★★★	★★★	★	★★
Marshalling Speed	★★	★★	★★★	★★★
Payload Size	★	★★	★★★	★★★
Total	6	7	7	8

Table 5.2.3: Comparison between the message formats.

We choose to use a novel message format. The reason for this is that it scores the highest and with a novel message format we reduce the size of each packet thereby preserving bandwidth for the client, which is in line with our scalability and cost goals described in Section 2.2.1.

### 5.2.4 Message Encryption

As explained in Section 3.1 we handle data where privacy is a concern. The communication between the clients and the server will be performed with HTTP, but it does not provide encryption of messages; this means that we need to find another way to encrypt them.

We look at two ways of encrypting traffic: Encrypting the message and encrypting only the body of the message.

Encryption of the message can be done with HTTPS [12]. HTTPS is HTTP over TLS [30], where TLS provides the secure connection.

Encryption of the body can be done with any available encryption algorithm.

We will now choose which type of encryption to use.

#### 5.2.4.1 Technology Choice

In Table 5.2.4 the comparison between the encryption methods can be seen.

	<b>HTTPS</b>	<b>Body Encryption</b>
Privacy	★★★	★★
Ease of use	★★★	★★
Total	6	4

Table 5.2.4: Comparison between the encryption methods.

We choose to use an HTTPS, because it encrypts all traffic and is easier to use.

## 5.3 Web Client

As stated in Section 2.2.1 the system should be able to showcase relevant information. Therefore, the purpose of the web client is to showcase information to drivers about their travel patterns, and what they paid for it. It should also allow the road pricing provider to see the billing information of drivers.

As stated in Chapter 4 the road pricing provider should not be able to see the travel patterns of the driver. This means that there should be a login functionality that distinguishes between a driver and the road pricing provider. It should not be possible for drivers to see the travel patterns or the billing information of any other driver.

No information should be shown if the user is not logged in. If a driver is logged in the billing information should be shown in a list. The travel information should be shown in a different view.

The road pricing provider should be able to submit a user id and view the billing information of that user.

We need to decide which technology we want to use for developing this web client. We consider PHP [21], JSF [44], and ASP.NET [39].

### 5.3.1 Technology Choices

We compare technologies on our experience with it, and how easy it is to set up. In Table 5.3.1 a comparison of the technologies are shown.

	<b>PHP</b>	<b>JSF</b>	<b>ASP.NET</b>
Experience	★★★	★★	★
Setup	★★★	★★	★★
Total	6	4	3

Table 5.3.1: Comparison between the web client technologies.

Based on the comparison table we choose PHP. The reason for choosing PHP is that it is a language we have experience with, we know that we can setup and create a quality web client in reasonable time.

## 5.4 Databases

This section analyzes which type of data storage is most appropriate for the different components of the system.

As described in the system definition in Chapter 4 the system should be able to store information about the driver on both the mobile client and the server.

In the following the different storage options will be presented.

## 5.4.1 Mobile Client Data Storage

As stated in Section 5.1 we develop an Android app, we need to decide which storage options to use.

In Android the following storage options exist [28]:

### Shared Preferences

Is mostly used to store persistent primitive data types i.e. booleans, floats, integers, longs, and strings. The data is stored in key-value pairs and is therefore best used for storing the preferences and settings of an app.

### Internal Storage

Is mostly used to store non-persistent data in an app. The stored data is private to the app and when the app is uninstalled the data will be removed. The advantage is that the app does not rely on external storage. The disadvantage is that the storage size depends on the mobile client and therefore can be very small. The stored data persists through reboot of the client.

### External Storage

Is used to store public data on the shared external storage of the mobile client. The advantage is that the data can be shared across the apps on the mobile client. The disadvantage is that the data can be unavailable if the external data is mounted on to a computer or removed from the mobile client.

### SQLite Databases

Is used to store structured data. The database will be private to the app and removed when uninstalling the app. The supported data types can be seen in Table 5.4.1.

Data Type	Value
Integer	Signed integer
Text	Text string
Real	Floating point
Blob	Blob of data, stored exactly as inputted

Table 5.4.1: The supported data types in SQLite [50].

The advantages of using SQLite databases is the possibility to store and execute structured queries. A disadvantage is that SQLite only offers a small number of data types.

### 5.4.1.1 Technology Choice

In Table 5.4.2 the comparison between the storage options of the mobile client can be seen.

	Shared Preferences	Internal Storage	External Storage	SQLite
Storage Limitation	★	★★	★★★	★★★
Data Types	★★	★	★	★★★
ACID	★	★	★	★★★
Queries	★	★	★	★★★
Total	5	5	6	12

Table 5.4.2: Comparison between the storage options.

We choose to use SQLite as the data storage of our mobile client, because the storage size is only restricted to the available space on the client, it has several data types, is ACID compliant, and supports queries. Furthermore, a lot of our data has spatial properties and SQLite has a geospatial extension called Spatialite [18]. This extension is necessary to be able to do client-side calculation as we propose in Section 3.2.

## 5.4.2 Server Databases

As stated in Section 3.1 we will be storing travel information and billing information on the server. We are working with geospatial data and we need to use a DBMS that supports it. In the previous section we chose SQLite for the mobile client, but SQLite lacks some of the characteristics of an enterprise DBMS, such as stored procedures and high concurrency [51]. We therefore consider the following DBMSs for the server databases:

### Microsoft SQL Server(Spatial) [40]

Is a relational DBMS, which is currently the 3<sup>rd</sup> most popular DBMS system [34]. It is a commercial DBMS.

### Oracle Database [45]

Is a relational DBMS, which is currently the 1<sup>st</sup> most popular DBMS system [34]. It is a commercial DBMS.

### PostgreSQL with PostGIS [48][47]

Is a object-relational DBMS, which is currently the 4<sup>th</sup> most popular DBMS system [34]. It is an object-oriented DBMS. It is not a commercial DBMS.

All the DBMSs have a wide set of geospatial functions.

### 5.4.2.1 Technology Choice

In Table 5.4.3 the comparison between the DBMSs can be seen.

	<b>MS SQL Server</b>	<b>Oracle Database</b>	<b>PostgreSQL</b>
Experience	★★	★	★★★
Total	2	1	3

Table 5.4.3: Comparison between the DBMS's.

We choose PostgreSQL as our DBMS, because it has a great community and we have experience with the DBMS.

## Data Usage

In this chapter we analyze how we can calculate the data usage of the traffic sent between the mobile client and server. We want to be able to make this calculation because of the goals we set in Section 2.2.1. First we have a goal that the operational cost should be low. This means that we need to lower the traffic between the mobile clients and the server as this will have a direct effect on the operational cost. Secondly we have a goal that the system should be scalable. This affects the traffic between the mobile clients and the server because more users generate more traffic. This means that if we can keep the communication to a minimum we can serve more mobile clients with the same hardware.

In Section 13.3 data usage calculations for data collected by test drivers will be performed.

In the following sections we introduce the method for calculating data usage for billing information and travel information.

### 6.1 Send Billing information

Billing information describes how much a driver should be paying in road pricing tax in a specific time interval. The billing information is sent to the server when a new time interval begins. The data that is being transferred every time is an identifier, a start time for the interval, a taxation model, and a price.

The size of the data sent for each report does not change based on report frequency. This means that the data cost of sending billings is linearly dependent on how frequent it is being sent.

We can calculate how much data a driver uses to send billing information in a day if we assume the following:

- The driver spends  $m$  minutes each day driving.
- The billing information is sent to the server with an interval of  $t$ .

- The size of the data sent between the client and the server when reporting a billing is  $d$ .

Using this notation we calculate the amount of billing information data sent per day as  $b$ :

$$b = \left\lceil \frac{m}{t} \right\rceil * d \quad (6.1)$$

## 6.2 Send Travel Information

The travel information describes the continuous location of the driver. The data is stored locally before it is transferred to the server. When the travel information is reported all data that has not previously been reported is sent to the server. This means that the size of the data sent is high if it is sent with low frequency, and low if it is sent with high frequency.

A new location is stored every time the driver enters a new segment. In order to calculate how much data a driver needs to send per day as travel information we need to make some assumptions:

- The driver spends  $m$  minutes each day driving.
- The driver enters a new road segment with a constant rate of  $u$ .
- The travel information is sent to the server with an interval of  $t$ .
- When a driver enters a new road segment the size of the location update is  $s$ .
- The travel information contains identification  $c$ .
- The size of the data sent between the client and the server when reporting travel logs is  $d$ .

Using this notation we calculate the amount of travel log data sent per day as  $l$ :

$$l = \left\lceil \frac{m}{t} \right\rceil * (d + c) + \left\lceil \frac{m}{u} \right\rceil * s \quad (6.2)$$

# **Part II**

# **Design**





# System Architecture

In this chapter the system architecture and all of its components are presented. The system architecture has been derived from the system definition, which can be seen in Chapter 4. It consists of the following components: Web Client, Mobile Client with Local Database, Server, Taxation Model Database, Billing Database, Encrypted Travel Log Database, and User Database. The system architecture can be seen in Figure 7.0.1. In the figure the dashed lines represent data flow between the components. The cylinders represent databases and the box represents the server. The desktop PC represents a client with a web browser and the smartphone represents a smartphone.

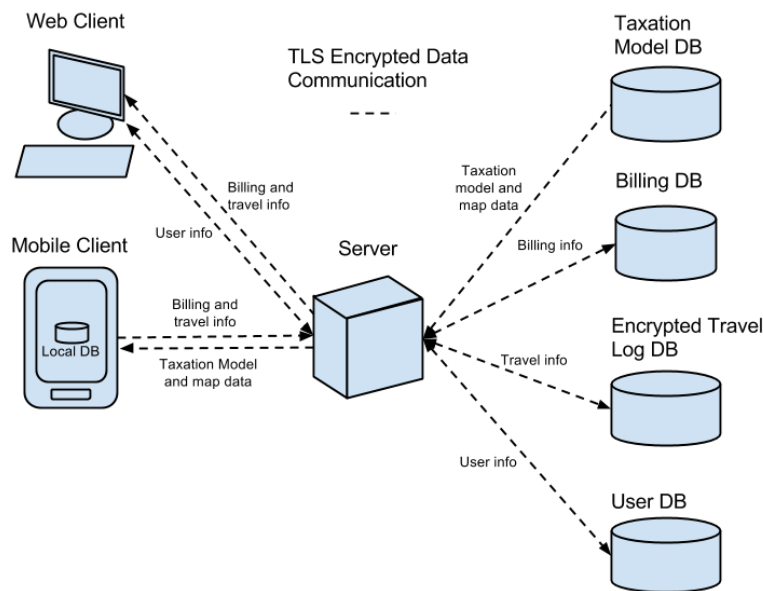


Figure 7.0.1: The figure shows all the components that make up the system architecture, and it shows the data flow between the components.

In the following sections the individual components will be introduced. For each component a flowchart [15] is presented. The charts provide an overview of the process and data flow. In the flowcharts we use the following symbols:

- A square represents a process.
- A diamond represents a decision.
- A parallelogram represents data.
- A cylinder represents data stored in a database.
- An arrow shows the direction of the data flow.

## 7.1 Mobile Client

The mobile client is the main source of data because it collects GPS information. This data is used to calculate the price of a trip, which is stored in travel logs. These travel logs are sent to the server in bulk.

The price calculations are aggregated over time and sent as billing information to the server without any location data. This means that the billing information cannot be used to track the driver, which means that the whereabouts of the drivers are inaccessible to road pricing providers. The travel logs contain segments derived from map matching. This means that the travel logs can be used to track the whereabouts of the driver. The billings are used to tax the driver, and travel logs are used to determine the whereabouts of the driver.

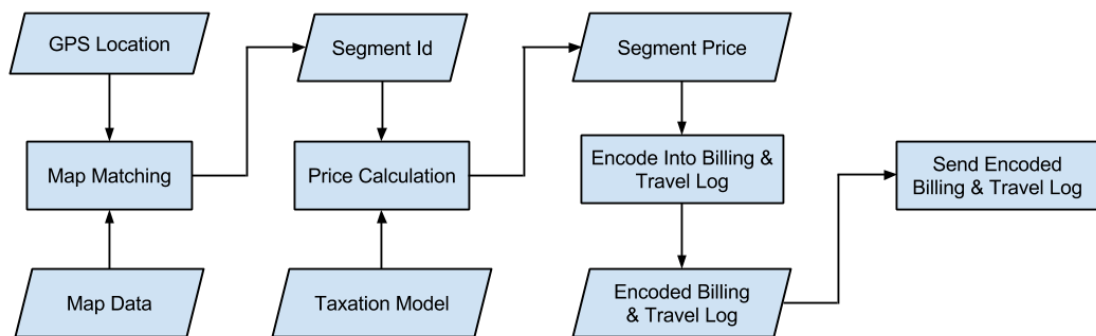


Figure 7.1.1: The data flow for the data created and processed by the mobile client component.

The process and data flow for map matching and price calculation on the mobile client component can be seen in Figure 7.1.1. First the GPS location for the mobile client is found. The GPS location and a map is used as input for the map matching function. The map matching function finds a segment id based on the input. When a segment id has been found, the price calculation function can calculate the price based on this id

and the active taxation model. Finally, the mobile client encodes the data as travel logs and billings and sends them to the server.

The process and data flow for getting the active taxation model and map data can be seen in Figure 7.1.2. At first the mobile client downloads the active taxation model id and the map data id. These ids are used to verify if the locally stored taxation model is the active one and if the map data is available on the mobile client. If one of these are invalid the mobile client will download the missing data from the server.

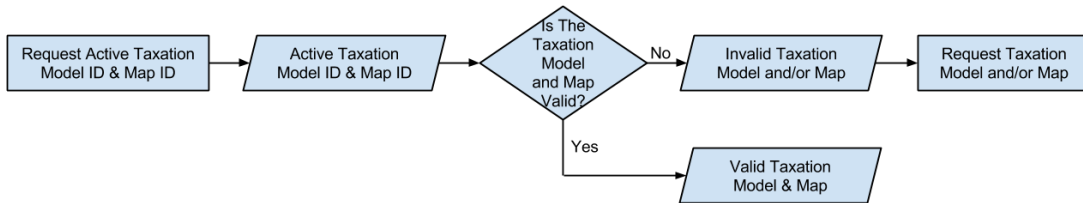


Figure 7.1.2: The data flow for the data fetched by the mobile client.

### 7.1.1 Local Database

The local database stores the following information: the whereabouts of the drivers, the active taxation model, relevant map data, and the calculated billing information.

The whereabouts of the drivers are stored to show where they have driven.

The active taxation model is stored in order to provide a basis for calculating the price for driving on a road segment.

The map data in the local database is used for map matching.

The calculated billing information is stored on the mobile client and aggregated in order to conserve bandwidth.

## 7.2 Server

The server is the center of the architecture, this is because the server handles the information exchange between the clients and databases.

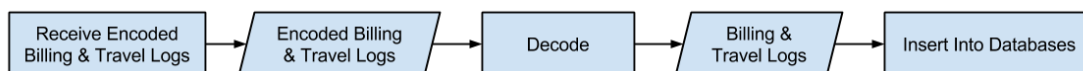


Figure 7.2.1: The data flow for incoming data in the server component.

The process and data flow for the incoming data can be seen in Figure 7.2.1. The server receives data from the mobile client in the form of billing information and travel

logs. These have been encoded in a novel message format in order to conserve bandwidth; therefore the server has to decode the data. After this is completed the data can be sent to the appropriate database for insertion.

The process and data flow for the outgoing data can be seen in Figure 7.2.2. The data flow for the outgoing data is very generic because the processes are very similar. The following data can be requested from the server: map data, taxation models, billing information, travel logs, and user information. The requested data is retrieved from the appropriate database and sent to the client that made the request.

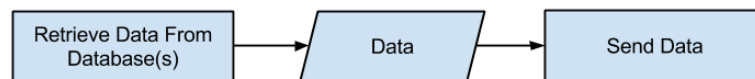


Figure 7.2.2: The data flow for outgoing data in the server component.

### 7.3 Web Client

The web client is intended both for the drivers who use the mobile client and the road pricing provider who provides the system, as well as audit users whose role is to inspect data of drivers. The web client provides a way of getting access to information that is stored in the system. The information comes from the mobile client and is stored in the appropriate databases.

The process and the data flow for the web client can be seen in Figure 7.3.1. The web client requests the needed billing information and travel logs from the server. The web client transforms travel and billing data before it is displayed to make it more readable.



Figure 7.3.1: The data flow for web client component.

### 7.4 Taxation Model Database

The taxation model database contains the information provided by the road pricing provider. The information in this database comes from RCS [41], which is a system that handles taxation models and map data. A taxation model defines a set of spatiotemporal rules that can be used for road pricing [41, pp. 19-25].

The process and data flow for the taxation model database can be seen in Figure 7.4.1. The database handles requests from the server and sends the result data back to the server.

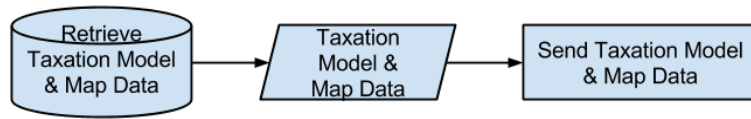


Figure 7.4.1: The data flow for the taxation model database component.

## 7.5 Billing Database

The billing database contains the information that is used by the road pricing provider to bill drivers. The information that is needed to bill a driver is: identification, time, taxation model, and price. The combination of these make up the billing information in the system.

- The identification is used to identify the driver that should be billed.
- The time describes the interval that the cost is generated in.
- The taxation model is used to verify that the driver is paying the correct road pricing tax.
- The price is an aggregated price for the segments driven.

The process and data flow for when the billing database receives billing information can be seen in Figure 7.5.1. The database receives data from the server. This data is inserted into the billing table, which stores the billing information.

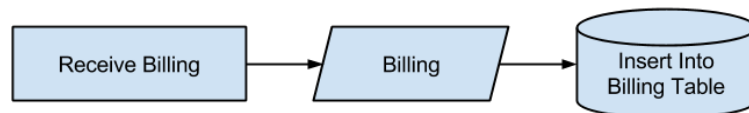


Figure 7.5.1: The data flow for the billing database component when it receives data.

The process and data flow for when the billing database retrieves billing information can be seen in Figure 7.5.2. The database handles requests for billings from the server. The requested billings are read from the database and returned to the server.

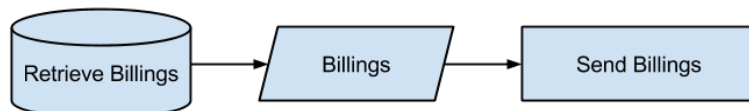


Figure 7.5.2: The data flow for the billing database components retrieval of data.

## 7.6 Encrypted Travel Log Database

This database contains the information about the whereabouts of the driver. It is necessary to store this information because it allows the audit user to verify the billing information. The information that is needed to verify the billing information is: identification, time, segment, taxation model, and price. The combination of these make up the travel logs in the system.

- The identification is used to identify the driver that should be billed.
- The time is used to determine when the driver is at a specific segment.
- The segment identifies the whereabouts of the driver.
- The taxation model is used to verify that the driver is paying the correct road pricing tax.
- The price is the calculated price for the segment.

The process and data flow for when the encrypted travel log database receives travel logs can be seen in Figure 7.6.1. The database receives data from the server. The data is inserted into the encrypted travel log table, which stores the encrypted travel logs.

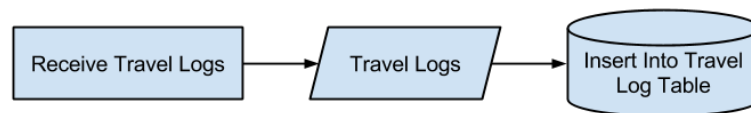


Figure 7.6.1: The data flow for the encrypted travel log database component when it receives data.

The process and data flow for when the encrypted travel log database retrieves travel logs can be seen in Figure 7.6.2. The database handles encrypted travel log requests from the server. This is handled by reading the requested data from the database and returning the result to the server.

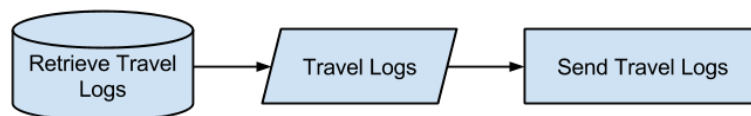


Figure 7.6.2: The data flow for the encrypted travel log database components retrieval of data.

## 7.7 User Database

The user database contains the information that is used by the road pricing provider to identify users. The information that is needed to identify a user is: identification,

username, password, and role. The passwords are hashed before they are stored in the database. The combination of these make up the user information in the system.

- The identification is used to identify the driver that should be billed.
- The username and password are used by the driver to access the web application.
- The role is used to distinguish between the groups of users in the system.

The process and data flow for when the user database receives user information can be seen in Figure 7.7.1. The database receives data from the server. This data is inserted into the person table, which stores the user information.

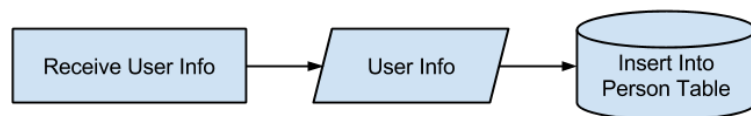


Figure 7.7.1: The data flow for the user database component when it receives data.

The process and data flow for when the user database retrieves user information can be seen in Figure 7.7.2. The database handles requests for user information from the server. The requested users are read from the database and returned to the server.

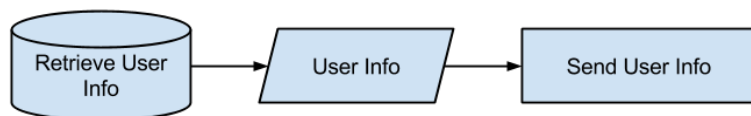


Figure 7.7.2: The data flow for the user database components retrieval of data.





## Client GUIs

In this chapter the design of the GUIs for the mobile app and web application are presented. The development of the mobile app and web application is focused on functionality. This means that the main concern is that they functions correctly, and usability is a secondary concern. The implementation of the mobile app and web application can be seen in Chapter 10 and Chapter 11 respectively.

### 8.1 Mobile App

The app is designed with two purposes in mind: showcasing road pricing tax to the driver and calculating road pricing tax. Continuously showcasing the road pricing tax is done so the drivers will get an immediate understanding of how road pricing will affect them. The app should also calculate the road pricing tax to limit data usage and lower response time, as described in Section 3.2.

We divide the app into four screens: Main menu, taximeter, settings, and travel history. The screens and the navigation between them are described in the following sections.

#### Main Menu

The main menu presents the driver with a set of options. The driver can start road pricing, see travel history, or go to the settings:

##### **Start Road Pricing**

This button will take the driver to the taximeter screen. Additionally, it will start the road pricing tax calculation.

##### **Travel History**

This button will take the driver to the travel history screen. In the screen the driver can see his travel history.

## Settings

This button will take the driver to the settings screen. Here the driver can change the available settings.

In Figure 8.1.1 the design for the main menu is shown.

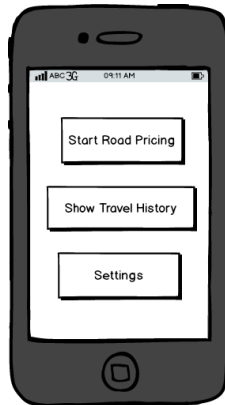


Figure 8.1.1: The design for the main menu screen of the app.

## Taximeter

The taximeter screen is where the driver can see the price for the current trip. This screen is available when the road pricing tax of a trip is being calculated. The design of this screen is very minimalistic due to the fact that it is presented to the driver while driving. The price on the screen should update as the driver progresses on his trip just as a taximeter in a taxi would, which also means the price will only increase. The taximeter screen can be seen in Figure 8.1.2.

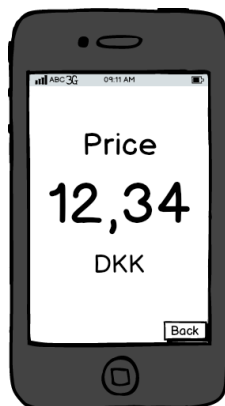


Figure 8.1.2: The design for the taximeter screen of the app.

## Travel History

The travel history screen shows the driver information about previous trips. The design for the travel history screen can be seen in Figure 8.1.3.

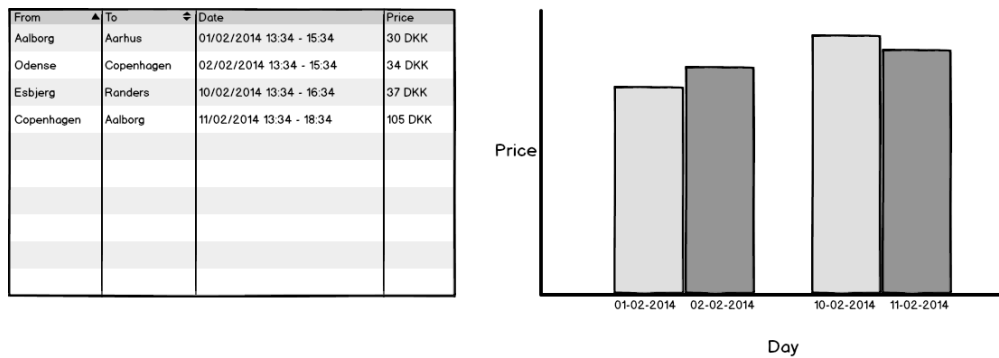


Figure 8.1.3: The design for the travel history screen of the app.

## Settings

The settings screen is where the driver is presented with the mobile app settings. This screen should be available in two versions depending on the user.

The driver should have a simplistic version where only the most basic settings are presented. The settings that are available for the driver are: Sync options, manual upload travel history, and manual update taxation model. The sync options are available to let the driver conserve the mobile bandwidth on the mobile client. The manual upload button allows the driver to upload the travel history before the mobile app would have uploaded it to the server. This may make the travel history available on the web application sooner than it would otherwise have. The manual update button allows the driver to download a new taxation model sooner than it had been scheduled for download. This will not activate the new taxation model but it will make it available on the mobile client.

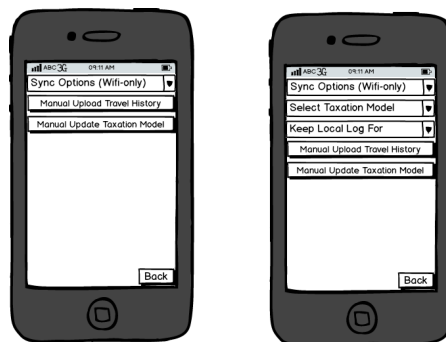


Figure 8.1.4: The design for the settings screen of the app.

The road pricing provider will have more settings available. One of the settings that are available are select taxation model. The other setting is how long the travel log should be available on the mobile client.

Both versions of the settings screen can be seen in Figure 8.1.4.

## Navigation

The navigation of the mobile app has its basis in the main menu where the user starts. From this screen the user can select any one of the other screens. From all other screens it is possible to navigate back to the main menu. The screens are shown in Figure 8.1.5 with navigational arrows.

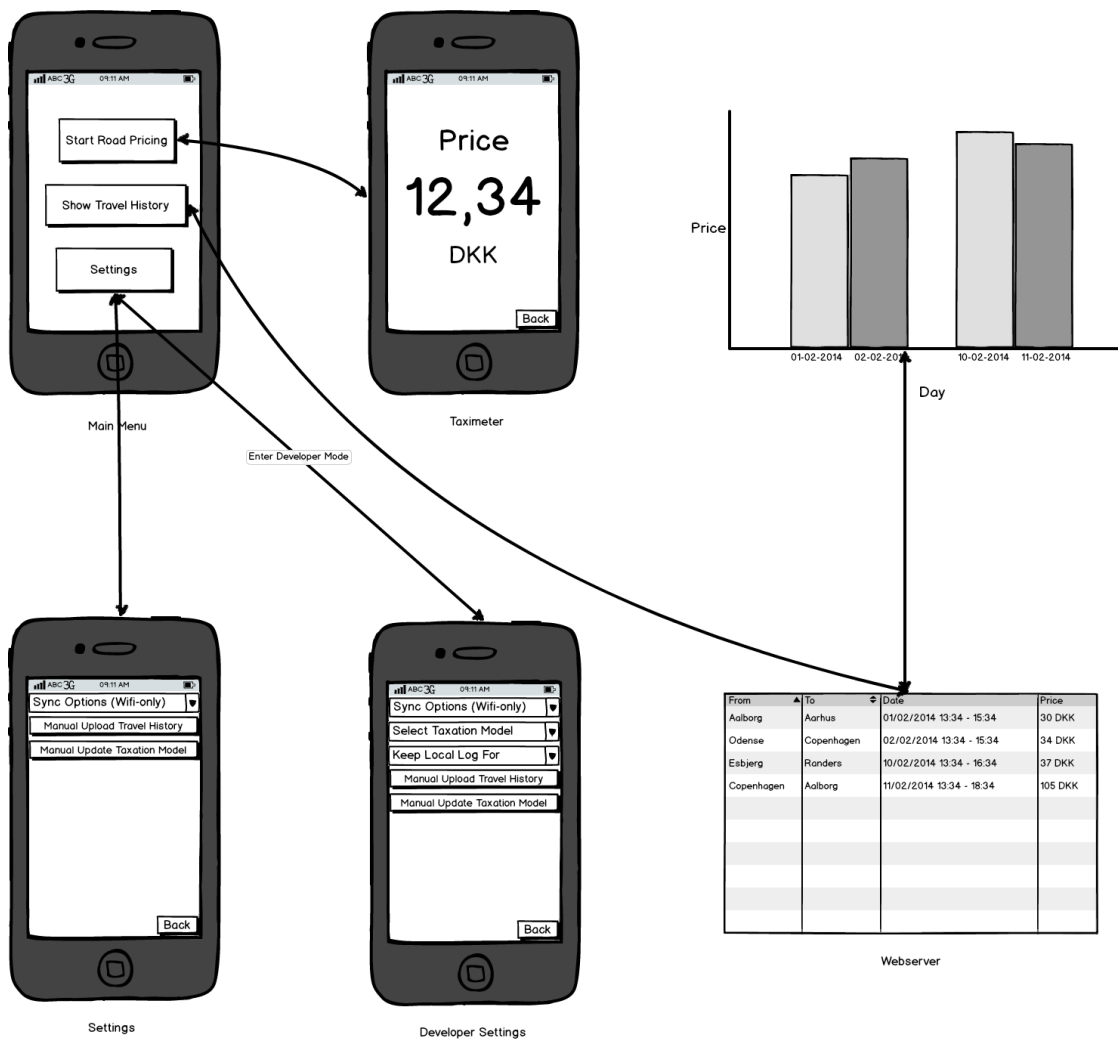


Figure 8.1.5: The navigation design for the app.

## 8.2 Web Application

The web application is only used for showcasing information to the users. This is in line with the showcasing goal described in Section 2.2.1. There are the following groups of users can use the web application: drivers, road pricing providers, and audit users. These groups of users have different access rights in the web application. The pages of the web application will be described below.

### Login

This page is a simple login screen with a login form consisting of user identification and password fields. If a user enters correct login information the user is redirected to the billing page. In Figure 8.2.1 the design for the login screen can be seen.

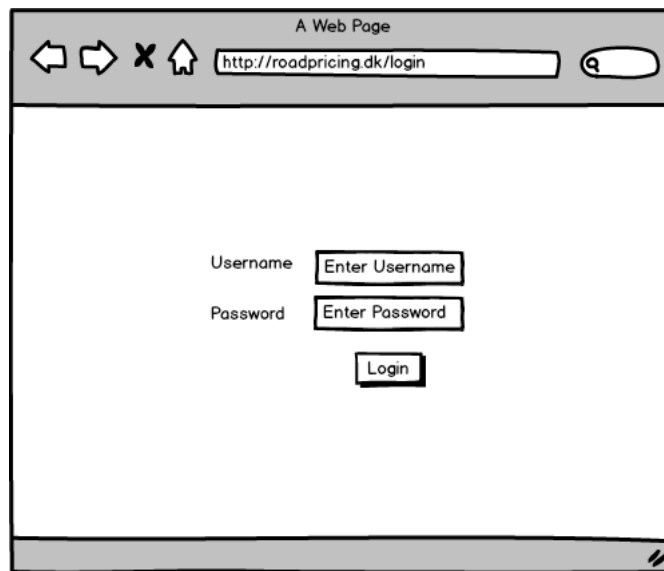


Figure 8.2.1: The design for login page for the web application.

### Billing

The billing page shows the billings the mobile client of the driver has sent to the server. The billing page is structured as a list of time intervals with corresponding amount of road pricing tax. If the user is logged in as a driver only the billing information of the user is displayed. If the user is logged in as a road pricing provider the user can choose to show the billing information of a single driver or statistics for all the drivers.

In Figure 8.2.2 an example of the design for the billing page can be seen. The figure shows both the page for a single driver's billing information and statistics for all the drivers.

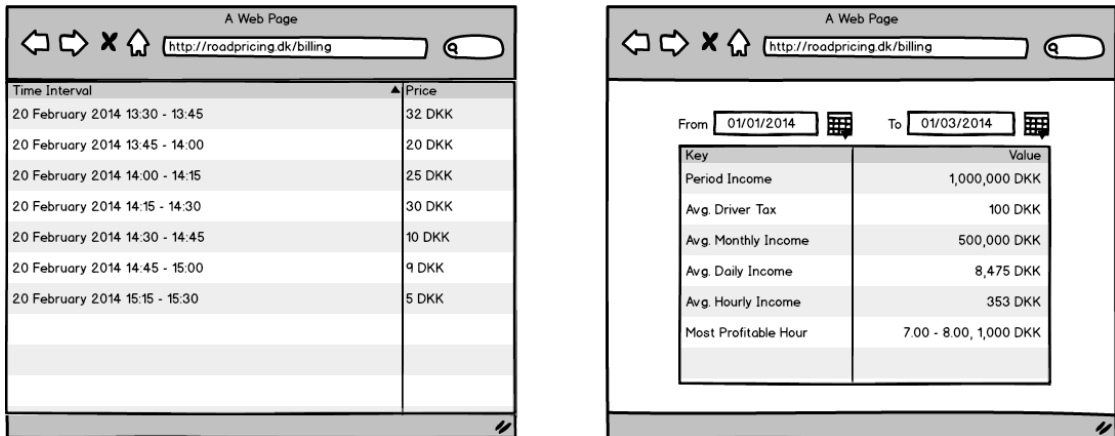


Figure 8.2.2: The design for billing page for the web application.

## Travel Log

This page shows precise travel information of a driver, and is therefore only accessible to the driver and audit user. If a user is logged in as a driver travel information is shown as a list. Each item in the list consists of a timestamp, location as a road segment, road pricing tax for the segment, length of the segment, and what was paid for driving on the segment.

In Figure 8.2.3 an example of the travel log page can be seen.

The image shows a browser window mockup for a travel log page. The title bar is 'A Web Page' and the URL bar contains 'http://roadpricing.dk/travellog'.

The table displays the following data:

Time Interval	Road	Seg Price	Seg Length	Price
20/02/2014 13:33 - 13:34	Køge Bugt Motorvejen	2	3	6
20/02/2014 13:34 - 13:35	Køge Bugt Motorvejen	2	4	8
20/02/2014 13:35 - 13:36	Køge Bugt Motorvejen	2	2	4
20/02/2014 13:36 - 13:37	Køge Bugt Motorvejen	2	2.5	5
20/02/2014 13:37 - 13:38	Køge Bugt Motorvejen	2	3.5	7
20/02/2014 13:38 - 13:39	Køge Bugt Motorvejen	2	1	2

Figure 8.2.3: The design for the travel log page for the web application.

## Database Structures

As explained in the system architecture in Chapter 7 the system contains the following databases: *Taxation Model Database*, *Billing Database*, *Encrypted Travel Log Database*, and *Local Database*.

In this chapter the databases and their schema diagrams will be presented.

### 9.1 Taxation Model Database

The *Taxation Model Database* is used to store taxation models and map data. An introduction to the database is given in Section 7.4. The schema for this database was developed for RCS [41, p. 31], which is a road pricing calculation system. The schema can be seen in Appendix A. It consists of the following tables: *map*, *model*, *area*, *model\_area\_group*, *rule*, *price*, and *temporal*.

A taxation model has a set of areas that have a set of rules. The schema supports any number of taxation models with any number of areas with any number of rules. In relation to RCS we have chosen to extend the *model* table with an *active\_from* attribute that defines the date a taxation model is active from.

### 9.2 Billing Database

The *Billing Database* is used by the road pricing provider to store the billing information the drivers provide. The database and the data it stores are introduced in Section 7.5. The schema for the *Billing Database* can be seen in Figure 9.2.1. It consists of the following table: *billing\_info*.

The table and its attributes will be explained in the following section.



billing_info		
PK	<u>imei</u>	BIGINT
PK	<u>time_start</u>	TIMESTAMP
	tm_id	UUID
	price	INTEGER

Figure 9.2.1: Database schema for the *Billing Database*.

### 9.2.1 billing\_info

The *billing\_info* table is used to store billing information, which is introduced in Section 7.5. The table consists of the following attributes: *imei*, *time\_start*, *tm\_id*, and *price*.

- *imei* is an identifier, which is used to store the IMEI [1] number of the mobile clients.
- *time\_start* describes the start of the time interval in the billing information.
- *tm\_id* identifies the taxation model of the billing information.
- *price* describes the price of the billing information.

In the *billing\_info* table we choose the attributes *imei* and *time\_start* as the primary key. This primary key enables us to uniquely identify each billing in the table. None of the attribute values can be null, because all attributes are needed to bill a driver.

## 9.3 Encrypted Travel Log Database

The *Encrypted Travel Log Database* is used to store the detailed travel information of all the mobile clients in the road pricing system. It consists of the *encrypted\_travel\_log* table that is encrypted. This is done to ensure the privacy of the mobile clients. The database and the data that it stores are introduced in Section 7.6. The schema for the *Encrypted Travel Log Database* can be seen in Figure 9.3.1.

encrypted_travel_info		
PK	<u>imei</u>	BIGINT
PK	<u>timestamp</u>	TIMESTAMP
PK	<u>segment_id</u>	INTEGER
	tm_id	UUID
	price	INTEGER

Figure 9.3.1: Database schema for the *Encrypted Travel Log Database*.

In the following section the *encrypted\_travel\_log* table is explained.

### 9.3.1 encrypted\_travel\_log

The *encrypted\_travel\_log* table is where the travel information of the drivers are stored. It consists of the following attributes: *imei*, *timestamp*, *segment\_id*, *tm\_id*, and *price*.

- *imei* is used to store the IMEI number of the mobile client.
- *timestamp* stores the time the driver drove on a given segment.
- *segment\_id* represents the segment the driver drove on.
- *tm\_id* identifies the taxation model the mobile client uses.
- *price* describes the cost of traveling on a given segment.

In the *encrypted\_travel\_log* table we have chosen the attributes *imei*, *timestamp*, and *segment\_id* as the primary key. This primary key enables us to uniquely identify each travel log in the table. The attribute values cannot be null.

## 9.4 User Database

The *User Database* is used by the road pricing provider to store the user information of the drivers. The database and the data it stores are introduced in Section 7.7. The schema for the *User Database* can be seen in Figure 9.4.1. It consists of the following table: *person*.

person		
PK	<u>username</u>	VARCHAR(128)
UK1	password imei role	VARCHAR(128) BIGINT SMALLINT

Figure 9.4.1: Database schema for the *User Database*.

### 9.4.1 person

The *person* table is used to store user information in the road pricing system. The user information for the drivers is used to link them with their mobile client. The table consists of the following attributes: *imei*, *username*, *password*, and *role*.

- *imei* is an identifier used to store the IMEI number of the mobile clients.
- *username* and *password* are used to validate users.
- *role* is used to distinguish between the user groups of the system.

In the *person* table we chose the attribute *username* as the primary key. Besides that the attribute *imei* is a unique key, which means that two *imei* values cannot be the same. The *imei* value can be null as this is only used to identify a drivers mobile client.

## 9.5 Local Database

The *Local Database* is used by the mobile clients to store the travel information of the drivers, the taxation model, and map data. The database and the data that it stores are introduced in Section 7.1.1. The schema for the *Local Database* is the combination of the schemas for the *Encrypted Travel Log Database* and *Taxation Model Database*. It consists of the following tables: *travel\_log*, *map\_data*, *taxation\_model*, *area*, *model\_area\_group*, *rule*, *price*, and *temporal*.

The *travel\_log* table is equivalent to the table from the *Encrypted Travel Log Database* in Section 9.3. But the *travel\_log* table on the mobile client is not encrypted. The rest of the tables are equivalent to the tables in the *Taxation Model Database*, which can be seen in Section 9.1.

# **Part III**

## **Implementation**



# Chapter 10

## Mobile App

This chapter explains the implementation of the Android mobile app. The app is designed to show the effects of road pricing online. This means that the app calculates and showcases the accumulated to the driver as he drives. The design used for the implementation of the app can be seen in Section 8.1.

In the following sections the main components of the app will be presented.

### 10.1 Activities

Activities [25] in Android is where interaction with the user is made. Each activity has its own window where it can setup its user interface. The app has the following activities: *SplashActivity*, *MenuActivity*, *TaxameterActivity*, *HistoryActivity*, and *SettingsActivity*. It is possible to start the *UpdateService* from all activities. This service handles the price calculations and will be presented in Section 10.2.1. In the following subsections each activity along with its responsibilities will be presented.

#### 10.1.1 SplashActivity

The *SplashActivity* is the first activity to be shown when the app is started. The window can be seen in Figure 10.1.1.

The activity downloads the *ApplicationInfo* through the web service. The *ApplicationInfo* consists of information of where to download the map and taxation model databases and the active taxation model. The map and taxation model databases will only be downloaded once. If the app cannot get internet connection it checks if it has already downloaded the databases and if a active taxation model exists. When the downloads are completed the activity directs the user to the *MenuActivity*.

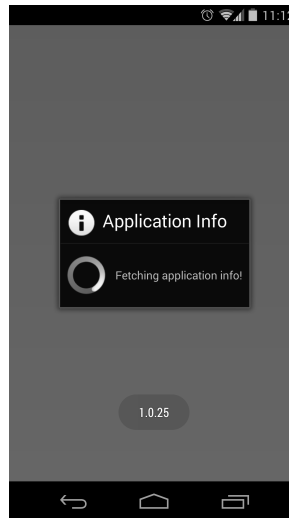
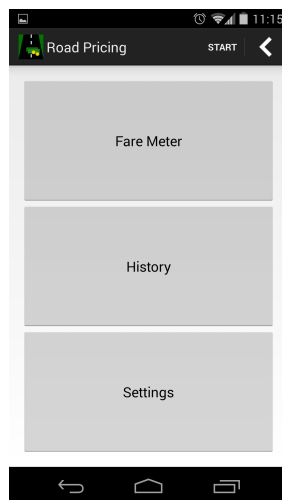


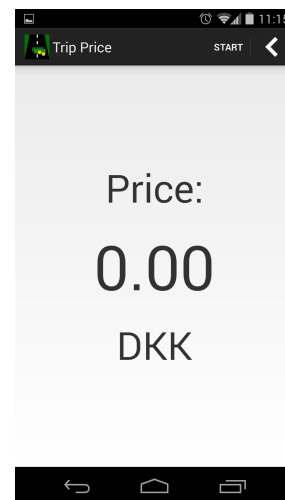
Figure 10.1.1: SplashActivity window.

### 10.1.2 MenuActivity

In the *MenuActivity* the user can navigate to *TaxameterActivity*, *HistoryActivity*, and *SettingsActivity*. This is accomplished through a simple button menu layout. The window for the activity can be seen in Figure 10.1.2a.



(a) MenuActivity window.



(b) TaxameterActivity window.

### 10.1.3 TaxameterActivity

The *TaxameterActivity* informs the user of the estimated cost of his current trip. This is done by a thread that inquires on the cost every 30<sup>th</sup> second from the *UpdateService*,

described in Section 10.2.1. The window for the activity can be seen in Figure 10.1.2b.

### 10.1.4 HistoryActivity

The *HistoryActivity* shows the 100 last visited segments of the user along with a timestamp and cost of each segment. This activity exists to give the user a simple way to see the segments used for estimating the cost. The window for the activity can be seen in Figure 10.1.2.

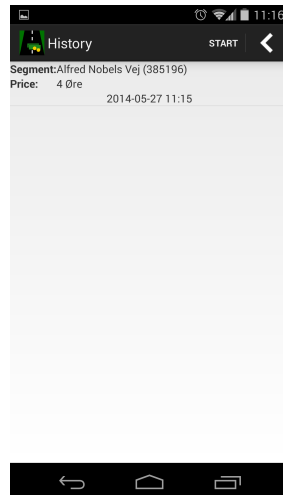


Figure 10.1.2: HistoryActivity window.

### 10.1.5 SettingsActivity

The *SettingsActivity* is where the billing and travel log communication intervals between the mobile client and the server are set. Furthermore, the IMEI number, the name of the active taxation model, and the battery information of the mobile client can be seen. This information is primarily used for testing purposes.

The activity also has a button that sends missing travel logs to the server, and a button that clears the history of the mobile client. The need for the missing travel logs button will be further explained in Section 10.2.2. The window for the activity can be seen in Figure 10.1.3.

## 10.2 Services

Services [29] in Android are run in the background in their own thread. There can only exist one instance of a service. The app consists of the following services: *UpdateService* and *WebServiceIntentService*. In the following subsections each service will be described.



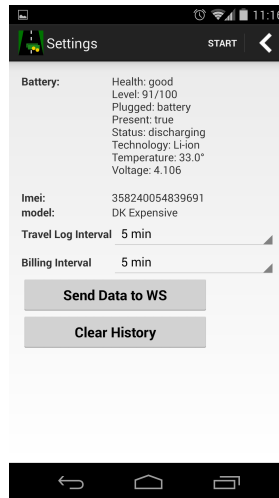


Figure 10.1.3: SettingsActivity window.

### 10.2.1 UpdateService

The *UpdateService* is a service that is created when the user starts a trip and runs until the trip is stopped. When the service is created it registers a location listener that receives GPS locations if the mobile client has moved 10 meters and the last received GPS location is older than one second.

Every time the service receives a new GPS location the *onLocationChanged* method is called. The code for this method can be seen in Code Snippet 10.2.1. The method starts by creating a new thread, which will be used to perform the database lookups. This can be seen in Section 10.2.1.

As seen in Section 10.2.1 the thread starts with finding the segment that corresponds to the received location. This is where map matching is performed, which will be explained in Section 10.3.1.

In Section 10.2.1 to Section 10.2.1 the thread checks if the segment has been visited recently. This is done to ensure that the driver does not pay for driving on the same segment twice. The recently visited segments is a list of the last 10 visited segments. If the segment was not visited recently the thread finds the price for the segment and adds it to the current trip cost. This can be seen in Sections 10.2.1 and 10.2.1.

The method call for finding the price of the segment will be explained in Section 10.3.2. This call is followed by sending the price of the segment and the segment to the *WebServiceIntentService* queue as seen in Section 10.2.1 to Section 10.2.1. The *WebServiceIntentService* will be presented in the following section.

```

1 @Override
2 public void onLocationChanged(Location location) {
3     final LatLng loc = new LatLng(location.getTime(), location.
4         getLatitude(), location.getLongitude());
5     new Thread(new Runnable() {
6         @Override
7         public void run() {
8             ... // Snip
9             Segment s = DBMap.getInstance(UpdateService.this).findSegment
10                (loc);
11             if (s != null) {
12                 boolean inPrevList = false;
13                 for (Segment prevS : prevSegments) {
14                     if (prevS.getSegmentID() == s.getSegmentID()) {
15                         inPrevList = true;
16                         break;
17                     }
18                 }
19                 if (!inPrevList) {
20                     prevSegments.add(0, s);
21
22                     if (prevSegments.size() > 10) {
23                         prevSegments.remove(prevSegments.size()-1);
24                     }
25
26                     int price = DBRp.getInstance(UpdateService.this).
27                         findSegmentPrice(s);
28                     taxameterPrice += price;
29
30                     Intent i = new Intent(UpdateService.this,
31                         WebServiceIntentService.class);
32                     i.putExtra(getString(R.string.billingPriceKey), price
33                         );
34                     i.putExtra(getString(R.string.segmentKey), s);
35                     startService(i);
36                 }
37             }
38             ... // Snip
39         }
40     }).start();
41 }

```

Code Snippet 10.2.1: The onLocationChanged method that is called when a new GPS location is received.

## 10.2.2 WebServiceIntentService

The *WebServiceIntentService* is a service that only exists when there are elements in its queue. It is in the *WebServiceIntentService* the billing of the user is handled. For each element in the queue the *WebServiceIntentService* calls the *onHandleIntent* method, which will check if the app needs to send data to the server.

The code for the *onHandleIntent* method can be seen in Code Snippet 10.2.2. The method starts by fetching the web service communication frequency for sending billings and travel logs. This can be seen in Sections 10.2.2 and 10.2.2. In Section 10.2.2 to Section 10.2.2 the current billing object is constructed. A billing object is not tied to a single trip. Instead it is the cost since the last time the app sent a billing object to the web service. In Sections 10.2.2 to 10.2.2 a travel log object is created and inserted in respectively the travel info table and history table. These tables will be further explained in Section 10.3.3.

If the mobile client is connected to the Internet, it checks if it should send the billing and travel logs. If this is the case the billing and the travel logs will be sent to the web service. This can be seen in Section 10.2.2 to Section 10.2.2.

```
1 @Override
2 protected void onHandleIntent(Intent intent) {
3     int price = intent.getIntExtra(getString(R.string.billingPriceKey),
4         0);
5     Segment s = intent.getParcelableExtra(getString(R.string.segmentKey))
6         ;
7     int bIntPos = settings.getInt(getString(R.string.
8         settings_billInterval), 0);
9     intervalBilling = getResources().getIntArray(R.array.billing_values)[
10        bIntPos];
11
12     int tIntPos = settings.getInt(getString(R.string.
13         settings_travelInterval), 0);
14     intervalTravel = getResources().getIntArray(R.array.travellog_values)
15        [tIntPos];
16
17     long curBTS = settings.getLong(getString(R.string.
18         current_billing_start), System.currentTimeMillis()/1000);
19     int curB = settings.getInt(getString(R.string.current_billing), 0);
20     int p = curB + price;
21     settings.edit().putInt(getString(R.string.current_billing), p).commit
22        ();
23     Billing billing = new Billing(Long.parseLong(imei), modelID, p,
24        curBTS);
25
26     TravelLog travel = new TravelLog(Long.parseLong(imei), modelID, price
27        , s.getTimestamp(), s.getSegmentID());
28     DBLocal.instance(this).insertTravelInfoTable(travel);
29     DBLocal.instance(this).insertHistoryTable(travel);
30
31     if (isNetworkAvailable()) {
32         long nowBilling = System.currentTimeMillis() - lastTimeBilling;
```

```

24     if (nowBilling > intervalBilling) {
25         lastTimeBilling = System.currentTimeMillis();
26         if (billing.getCost() > 0) {
27             sendBillings(billing);
28         }
29     }
30     long nowTravel = System.currentTimeMillis() - lastTimeTravel;
31     if (nowTravel > intervalTravel) {
32         lastTimeTravel = System.currentTimeMillis();
33         sendTravels();
34     }
35 }
36 ... // Snip
37 }

```

Code Snippet 10.2.2: The `onHandleIntent` method called for every element in the queue.

## 10.3 Databases

The app has the following SQLite databases: *map*, *road pricing*, and *localdb*. These databases are presented in Chapter 9. The app has one connection per database, because SQLite handles concurrency poorly. This is enforced through a singleton pattern as seen in Code Snippet 10.3.1.

```

1 private DBLocal(Context c) {
2     this.dbHelper = new DbHelper(c);
3     this.db = dbHelper.getWritableDatabase();
4 }
5
6 public static synchronized DBLocal instance(Context c){
7     if(instance == null){
8         instance = new DBLocal(c);
9     }
10    return instance;
11 }

```

Code Snippet 10.3.1: An example of a singleton creation of a database connection.

In the following subsections the important method calls for each database are presented.

### 10.3.1 map

The *map* database is where all the road segments are stored. The most commonly used method call to this database is *findSegment*, which can be seen in Code Snippet 10.3.2. It is a simple map matching algorithm that returns the segment closest to the GPS location.

It queries the *map* database to find the *id*, *streetname*, *source*, *target*, and *length* of a segment that corresponds to the given GPS location. The query can be seen in Section 10.3.1.

To speed up the query a minimum bounding rectangle is created. The minimum bounding rectangle is created with the dimensions: longitude - 0.02, latitude - 0.01, longitude + 0.02, latitude + 0.01.

```
1 public Segment findSegment(LatLng loc) {
2     Segment seg = null;
3     try {
4         int segmentid = 0;
5         String streetname = null;
6         int type = 0;
7         int source = 0;
8         int target = 0;
9         String magid = null;
10        double length = 0;
11
12        // Find segment id
13        String querySeg = "select id, streetname, roadtype, source,
14            target, GLength(transform(geom,32632)) from map m " +
15            "where st_intersects(buildmbr( " + (loc.getLng() - 0.02) + ",
16            " + (loc.getLat() - 0.01) + ", " + (loc.getLng() + 0.02)
17            + ", " + (loc.getLat() + 0.01) + "), m.geom) " +
18            "and m.ROWID in (" +
19            "select ROWID from SpatialIndex where f_table_name='map'
20            and search_frame=buildmbr( " + (loc.getLng() - 0.02) +
21            ", " + (loc.getLat() - 0.01) + ", " + (loc.getLng() +
22            0.02) + ", " + (loc.getLat() + 0.01) + ")) " +
23            "Order by distance(geom, MakePoint("+ loc.getLng() + ", "
24            + loc.getLat() + ", 4326)) limit 1;";
25
26        Stmt stmtSeg = db.prepare(querySeg);
27        if (stmtSeg.step()) {
28            segmentid = stmtSeg.column_int(0);
29            streetname = stmtSeg.column_string(1);
30            type = stmtSeg.column_int(2);
31            source = stmtSeg.column_int(3);
32            target = stmtSeg.column_int(4);
33            length = stmtSeg.column_double(5);
34        }
35
36        magid = DBRp.getInstance(context).findMAG(loc.getLat(), loc.
37            getLng());
38
39        seg = new Segment((loc.getTime()/1000), segmentid, streetname,
40            type, source, target, length, magid);
41    } catch (Exception e) {
42        e.printStackTrace();
43    }
44    return seg;
45 }
```

Code Snippet 10.3.2: The findSegment method.

### 10.3.2 road pricing

The *road pricing* database is where all the taxation models are stored. The most commonly used method call to this database is *findSegmentPrice*. It is equivalent to the stored procedure *find\_segment\_price* from [41, pp. 55-57]. The only difference is that the method has been converted to a Java method that calls a query. This is because SQLite does not support stored procedures.

### 10.3.3 localdb

The *localdb* database is where the travel logs and history of visited segments are stored. The database connection implements simple **C**reate, **R**ead, **U**ppdate, and **D**elede methods(CRUD).



## Web Application

As described in the showcasing goal in Section 2.2.1 there needs to be a way for drivers to see exactly what they are paying for. We therefore develop a web application that describes all temporal and spatial travel data associated with the road pricing tax for drivers. The implementation of the web application is based on the design described in Section 8.2. This web application is used by drivers, road pricing providers, and auditors.

Drivers use the web application to see what they should pay in road pricing tax along with an exact breakdown of what they should pay for each individual segment they traveled on.

Road pricing providers use the web application to see the overall revenue generated by road pricing as well as the revenue generated by the individual drivers. It is not possible for road pricing providers to see where a driver was – only what they owe for a user-defined time period.

Audit users are able to see where each individual driver has been and what he should pay in road pricing tax. The audit user should only be used if a specific driver is under reasonable suspicion of intentionally reporting falsified information.

### 11.1 Login

The GUI for the login page is identical for all users. When a user logs in with their username and password, their role is set. This role determines which content will be displayed. The login interface can be seen in Figure 11.1.1.



Figure 11.1.1: The login GUI for the web application.

## 11.2 Travel Log

The travel log page lists the price for each segment a driver has traveled on. The entries are divided into pages of 20. A driver will only see the prices for the segments he has traveled on, whereas an audit user can see this information for any driver. In Figure 11.2.1 the GUI for the travel log of a driver is shown. Because of our goal of keeping drivers' travel patterns private – as described Section 2.2.1 – road pricing providers are unable to see the travel log page. This means that road pricing providers are unable to find the location of a driver.

As described in the system architecture in Chapter 7 the travel logs are stored in a separate database. In the PHP implementation of the web application there is a distinct class that handles the connection to each separate database. This is used to easily handle the permissions of the user roles.

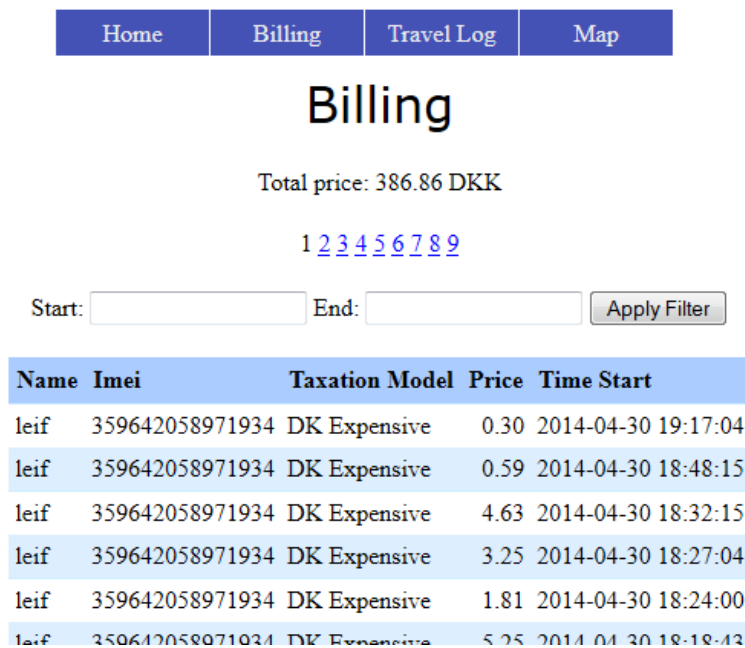
Name	Imei	Taxation Model	Price	Time Start	Segment
mikael	355577059649952	DK Expensive	0.09	2014-05-05 08:07:17	Sigrid Undsets Vej (152000)
mikael	355577059649952	DK Expensive	0.39	2014-05-05 06:04:09	Universitetsboulevarden (22670)
mikael	355577059649952	DK Expensive	0.39	2014-05-05 06:03:46	Universitetsboulevarden (22673)
mikael	355577059649952	DK Expensive	0.13	2014-05-05 06:03:04	Niels Bohrs Vej (376819)
mikael	355577059649952	DK Expensive	0.16	2014-05-04 16:26:46	Sigrid Undsets Vej (151998)
mikael	355577059649952	DK Expensive	0.82	2014-05-04 16:20:09	Selma Lagerlöfs Vej (380041)
mikael	355577059649952	DK Expensive	0.03	2014-05-04 16:01:26	Sigrid Undsets Vej (152002)

Figure 11.2.1: The travel log page for the driver 'mikael'.

## 11.3 Billing

The billing page displays the cost of driving in temporal intervals. Drivers are able to see all the billing information associated with them. Similar to the travel log page, the entries are divided into pages of 20. The billing page for a driver can be seen in Figure 11.3.1. Road pricing providers and audit users are able to see the billing information for any and all drivers.

The start time of the interval is displayed in the “Time Start” column. The price displayed is the cost of driving from the timestamp in the entry to the timestamp in the following entry.



The screenshot shows a navigation bar with four buttons: Home, Billing (selected), Travel Log, and Map. Below the navigation bar is the title "Billing" and the text "Total price: 386.86 DKK". There are pagination links "1 2 3 4 5 6 7 8 9" with "5" highlighted. Below the pagination are two input fields labeled "Start:" and "End:" followed by an "Apply Filter" button. The main content is a table with the following data:

Name	Imei	Taxation Model	Price	Time Start
leif	359642058971934	DK Expensive	0.30	2014-04-30 19:17:04
leif	359642058971934	DK Expensive	0.59	2014-04-30 18:48:15
leif	359642058971934	DK Expensive	4.63	2014-04-30 18:32:15
leif	359642058971934	DK Expensive	3.25	2014-04-30 18:27:04
leif	359642058971934	DK Expensive	1.81	2014-04-30 18:24:00
leif	359642058971934	DK Expensive	5.25	2014-04-30 18:18:43

Figure 11.3.1: The billing page for the driver 'leif'.

## 11.4 Map

In order to visualize the travel log data, it can be seen on an interactive map. The permissions for the map page are the same as the travel log: road pricing providers cannot access it. The map GUI of a driver can be seen in Figure 11.4.1.

## Map

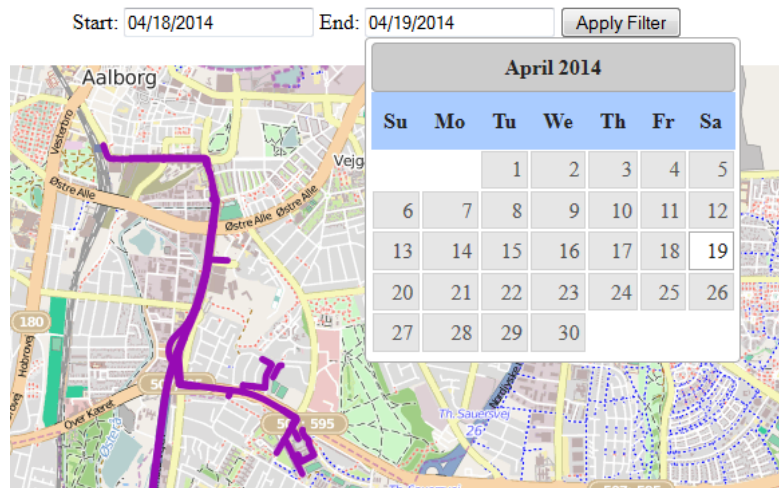


Figure 11.4.1: The map page for a driver with a temporal constraint applied.

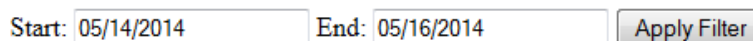


Figure 11.5.1: The filtering options drivers have for each page.

## 11.5 Filtering

Drivers are able to filter entries in the billing and travel log pages – as well as the highlighted segments in the map page – based on a user-defined temporal interval. The filtering form for drivers can be seen in Figure 11.5.1.

The road pricing providers and audit users are able to filter on a specific driver in addition to a temporal interval, which can be seen in Figure 11.5.2. The filtering options are the same for all pages.

The total price, which can be seen in Figure 11.2.1 and Figure 11.3.1, is also determined by the filter: If no filter is applied it will reflect the total price of the billings or travel logs. When a filter is applied only the entries that are within the filter are part of the aggregated price.

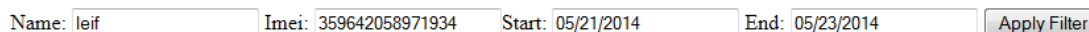


Figure 11.5.2: The filtering options road pricing providers and audit users have on the pages they have permissions to.

# Chapter 12

## Web Services

This chapter explains the implementation of the REST web services as designed in the data flows in Section 7.2. The server is responsible for hosting the REST web services in the system. We implement the web services to accommodate the needs of the mobile app. The following web services are needed: *getApplicationInfo*, *sendBilling*, and *sendTravelLog*. In the following sections each web service will be described.

### 12.1 getApplicationInfo

The *getApplicationInfo* web service is used by the mobile client to get information on the newest databases and the active taxation model. The implementation can be seen in Code Snippet 12.1.1.

*getApplicationInfo* is a HTTP GET web service that returns the *ApplicationInfo* object as a comma-separated string. The *ApplicationInfo* object consists of: the URL for downloading the latest *map* database, the URL for downloading the latest *road pricing* database, and the identifier for the active taxation model.

```
1 @RequestMapping(value = "/applicationinfo", method = RequestMethod.GET,
2   produces = "text/plain")
3 public String getApplicationInfo(){
4     ApplicationInfo ai = new ApplicationInfo(
5         "http://130.225.198.79:6060/RPServer/files/map.db",
6         "http://130.225.198.79:6060/RPServer/files/roadpricing.db",
7         "899b2ce2-ab5e-11e3-9c2f-00269edefa15",
8         ... // snip
9     );
10    return ai.convertToString();
}
```

Code Snippet 12.1.1: The *getApplicationInfo* web service.

## 12.2 sendBilling

The *sendBilling* web service is used by the mobile client to send *Billing* objects to the *billingdb*. The *billingdb* is described in Section 9.2 and it is used to store the *Billing* objects. The implementation of the *sendBilling* web service can be seen in Code Snippet 12.2.1.

```
1 @RequestMapping(value = "/billing", method = RequestMethod.POST, consumes
  = "*/*", produces = "text/plain")
2 public String sendBilling(@RequestBody byte[] input) {
3     Billing b = Billing.ConstructFromBytes(input);
4     return bm.insertBilling(b)?"1":"0";
5 }
```

Code Snippet 12.2.1: The *sendBilling* web service.

*sendBilling* is a HTTP POST web service and it takes a *Billing* object as input. This *Billing* object is in the form of a byte array. The structure of the *Billing* byte array can be seen Section 12.2.1. The *sendBilling* web service returns 1 if the *Billing* object is inserted into the *billingdb* or 0 if an error occurred.

### 12.2.1 Billing Message Format

The *Billing* object consists of the attributes described in Section 9.2.1. The *Billing* object always has the same size, because all the attributes are of a fixed size. Therefore the attributes are stored consecutively after each other. In Figure 12.2.1 the byte sizes of the attributes can be seen.

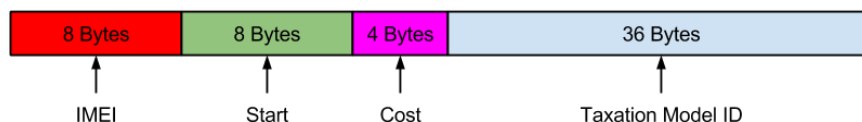


Figure 12.2.1: Structure of a *Billing* byte array.

The size of a *Billing* byte array is 56 bytes:

- Bytes 0 through 7 represent the IMEI number of the mobile client.
- Bytes 8 through 15 represent the Unix epoch time the *Billing* was created, called *start*.
- Bytes 16 through 19 represent the cost of the *Billing* object.
- Bytes 20 through 56 represent the taxation model identifier.

## 12.3 sendTravelLog

The *sendTravelLog* web service is used by the mobile client to send *TravelLog* objects to the *traveldb*. The *traveldb* is described in Section 9.3 and is used to store the *TravelLog* objects. The implementation of the *sendTravelLog* web service can be seen in Code Snippet 12.3.1.

```
1 @RequestMapping(value = "/travellog", method = RequestMethod.POST,
   consumes = "*/*", produces = "text/plain")
2 public String sendTravelLog(@RequestBody byte[] input) {
3     List<TravelLog> list = TravelLog.ConstructListFromBytes(input);
4     boolean allGood = tm.insertTravels(list);
5     return allGood?"1":"0";
6 }
```

Code Snippet 12.3.1: The sendTravelLog web service.

*sendTravelLog* is a HTTP POST web service and takes a list of *TravelLog* objects as input. This list is in the form of a byte array. The structure of the byte array can be seen Section 12.3.1. The *sendTravelLog* web service returns 1 if the list is inserted in the *traveldb* or 0 if an error occurs.

### 12.3.1 TravelLog Message Format

The *Travellog* object consists of the attributes described in Section 9.3.1. In Figure 12.3.1 the structure of a *TravelLog* byte array can be seen. The example in the figure consists of three *TravelLog* objects: two created when **model-1** was the active *Taxation Model* and one when **model-2** was the active *Taxation Model*. A concrete example of this structure can be seen in Section 13.2.

```
[imei]
  [model-1]
    [cost-1-a];[start-1-a];[segment-1-a]
    $
    [cost-1-b];[start-1-b];[segment-1-b]
  |
  [model-2]
    [cost-2-a];[start-2-a];[segment-2-a]
```

Figure 12.3.1: Structure of a list of Travellogs as byte array.

The size of the byte array depends on the number of *TravelLog* objects. The first 8 bytes always represent the IMEI number of the mobile client. The IMEI number is the same for all the *TravelLog* objects and is therefore only displayed once.

The rest of the bytes are divided into groups based on their *Taxation Model ID*. These groups are divided with the “|” symbol, which is colored red in the figure. The first

36 bytes of each group represent the *Taxation Model ID*. The remaining bytes in each group is represented as triples on the form *cost;start;segment*. Each of these triples are divided by a “\$” symbol, which is colored blue in the figure.

**Part IV**

**Evaluation**





# Chapter 13

## Results

In this Chapter the results of this project will be presented.

### 13.1 Field Trial

As presented in Section 2.2.1, accuracy is an important goal of our system. This is to ensure the drivers pay the correct road pricing tax. Because of this, a field trial was performed to evaluate the accuracy of the system, and to show that the road pricing system and all its components are fully operational.

#### 13.1.1 Overall Test Results

In the period April 8<sup>th</sup> to June 3<sup>rd</sup> 7 drivers have tested the road pricing system. Combined they have driven 4304 kilometers and generated 2736.9 DKK in travel logs and 2511.5 DKK in billings. The difference is caused by the different reporting frequencies of billings and travel logs. This means that some billings are stored locally on the mobile client and have not yet been reported to the server.

The travel log database contains 10013 rows, which each represent a travel log. The billing database contains 810 rows, which each represent a billing.

#### 13.1.2 Structured Test Drive

The structured test drive consists of three trips. The test drive was conducted using a LG Nexus 5 [22]. All trips have been performed on the same route of approximately 14 kilometers, which consists of 49 segments. The route can be seen in Figure 13.1.1.

For each trip we locate the mobile client differently in the car.

- In trip 1 the mobile client is located in the passenger seat of the car.
- In trip 2 the mobile client is located in the glove box of the car.
- In trip 3 the mobile client is located in the front window of the car.



Figure 13.1.1: The route of the test drive [42].

The results from the test drive can be seen in Table 13.1.1. From the results we can see that the number of segments detected is almost identical. Trip 3 misses three segments, but two of them are the start and stop segment, which could be attributed to the driver pressing the start/stop button too late. The location of the mobile client does not seem to interfere with the results.

Furthermore, we can see that the mobile app detects a lot of the side streets, making the price of the trip greater than the expected road pricing tax. This is caused by the simple map matching algorithm of the mobile app, which is too aggressive when it comes to finding segments. A discussion on map matching can be seen in Section 14.2.

	<b>Trip 1 - Passenger Seat</b>	<b>Trip 2 - Glove Box</b>	<b>Trip 3 - Front Window</b>
<b>Total</b>	62 segments	63 segments	60 segments
<b>Missed</b>	Loftbrovej(54123)	Loftbrovej(54123)	Blomsterparken(239654) Gammel Høvej(66907) Blomsterparken(239654)
<b>Correct</b>	48/49 segments (98%)	48/49 segments (98%)	46/49 segments (93.9%)
<b>Wrong</b>	14 segments (22.6%)	15 segments (23.8%)	14 segments (23.3%)

Table 13.1.1: Segment results of the test drive.

## 13.2 Message Size

As presented in Section 2.2.1 operational costs and scalability are important goals of our system. Therefore it is desirable to keep the HTTP messages sent between the mobile client and the server as small as possible. We compare our novel message format against XML and JSON.

### 13.2.1 Billing Comparison

The implementation of the billing message format can be seen in Section 12.2.1. For this comparison the following *Billing* object is used:

#### Billing 1

**IMEI:** 123456789012345  
**Model:** 3b1d50ed-ffe9-4966-aff0-63cf1a136625  
**Cost:** 123  
**Start:** 1401615421

In Code Snippet 13.2.1 the *Billing* object is transformed into our novel message format. The size of the message is 64 bytes, but after transformation it is shorter. This is because the int and longs are transformed to bytes reducing them to respectively 4 and 8 bytes. Therefore, the size of the message is 56 bytes.

```
12345678901234514016154211233b1d50ed-ffe9-4966-aff0-63cf1a136625
```

Code Snippet 13.2.1: The Billing in the novel message format before the int and longs are transformed.

The *Billing* object can be seen as XML in Code Snippet 13.2.2 and as JSON in Code Snippet 13.2.3.

```
<?xml version="1.0" ?>
<billing>
  <imei>123456789012345</imei>
  <model>3b1d50ed-ffe9-4966-aff0-63cf1a136625</model>
  <cost>123</cost>
  <start>1401549053544</start>
</billing>
```

Code Snippet 13.2.2: The Billing as XML.

```
{"imei":123456789012345,"model":"3b1d50ed-ffe9-4966-aff0-63cf1a136625","cost":123,"start":1401549053544}
```

Code Snippet 13.2.3: The Billing as JSON.

In Table 13.2.1 the comparison results between our novel message format, XML, and JSON can be seen. The size of the XML message is 161 bytes, which is 187.5% larger than our novel message format. The size of the JSON message is 101 bytes, which is 80.4% larger than our novel message format.

	<b>Novel Format</b>	<b>XML</b>	<b>JSON</b>
Message Size	56 bytes	161 bytes	101 bytes
% Larger	–	187.5%	80.4%

Table 13.2.1: Billing message comparison table.

### 13.2.2 TravelLog Comparison

The implementation of the travel log message format can be seen in Section 12.3.1. For this comparison the following *TravelLog* objects are used:

#### TravelLog 1

**IMEI:** 123456789012345  
**Model:** 3b1d50ed-ffe9-4966-aff0-63cf1a136625  
**Cost:** 111  
**Start:** 1401615421  
**Segment:** 11111

#### TravelLog 2

**IMEI:** 123456789012345  
**Model:** 3b1d50ed-ffe9-4966-aff0-63cf1a136625  
**Cost:** 222  
**Start:** 1401615421  
**Segment:** 22222

#### TravelLog 3

**IMEI:** 123456789012345  
**Model:** 979f8f93-bb7e-4486-bc00-54ba5aac5945  
**Cost:** 222  
**Start:** 1401615421  
**Segment:** 22222

In Code Snippet 13.2.4 the *TravelLog* objects are transformed into our novel message format. The size of the message is 158 bytes, but after transformation it is shorter. This is because only the IMEI number is shortened to 8 bytes. The size of the message is 142 bytes.

```
1234567890123453b1d50ed-ffe9-4966-aff0-63cf1a136625111;1401615421;11111
$222;1401615421;22222|979f8f93-bb7e-4486-bc00-54ba5aac5945222
;1401615421;22222
```

Code Snippet 13.2.4: The TravelLogs in the novel message format before the IMEI is transformed.

The *TravelLog* objects can be seen as XML in Code Snippet 13.2.5 and as JSON in Code Snippet 13.2.6.

```
<?xml version="1.0" ?>
<travelloglist>
  <travellog>
    <imei>123456789012345</imei>
    <model>3b1d50ed-ffe9-4966-aff0-63cf1a136625</model>
    <cost>111</cost>
    <start>1401615421</start>
    <segment>11111</segment>
  </travellog>
  <travellog>
    <imei>123456789012345</imei>
    <model>3b1d50ed-ffe9-4966-aff0-63cf1a136625</model>
    <cost>222</cost>
    <start>1401615421</start>
    <segment>22222</segment>
  </travellog>
  <travellog>
    <imei>123456789012345</imei>
    <model>979f8f93-bb7e-4486-bc00-54ba5aac5945</model>
    <cost>222</cost>
    <start>1401615421</start>
    <segment>22222</segment>
  </travellog>
</travelloglist>
```

Code Snippet 13.2.5: The TravelLogs as XML.

```
[
  {"imei":123456789012345,"model":"3b1d50ed-ffe9-4966-aff0-63cf1a136625",
  "cost":111,"start":1401615421,"segment":11111},
  {"imei":123456789012345,"model":"3b1d50ed-ffe9-4966-aff0-63cf1a136625",
  "cost":222,"start":1401615421,"segment":22222},
  {"imei":123456789012345,"model":"979f8f93-bb7e-4486-bc00-54ba5aac5945",
  "cost":222,"start":1401615421,"segment":22222}
]
```

Code Snippet 13.2.6: The TravelLogs as JSON.

In Table 13.2.2 the comparison results between our novel message format, XML, and JSON can be seen. The size of the XML message is 554 bytes, which is 290.1% larger than our novel message format. The size of the JSON message is 355 bytes, which is 150% larger than our novel message format.

	Novel Format	XML	JSON
Message Length	142 bytes	554 bytes	355 bytes
% Larger	–	290.1%	150%

Table 13.2.2: TravelLog message comparison table.

## 13.3 Data Usage

As presented in Section 2.2.1, operational costs and scalability are important goals of the system. This is to ensure that the cost of running the system is low compared to the generated revenue and that we can handle millions of users. Therefore, we want to calculate the amount of data transferred between the mobile client and the server.

In Chapter 6 we analyze how to calculate the data usage. With the implemented system we record packages transferred between the mobile client and the server in order to calculate the data usage. We want to calculate the actual data usage using both HTTP and HTTPS. This will illustrate the cost of using encryption.

### 13.3.1 Daily Data Usage of a Single Driver

We calculate the data usage for sending a billing and travel logs using both HTTP and HTTPS. In Figure 13.3.1 the packages sent between the mobile client and the server when reporting a billing using HTTP is shown. In Appendix C the packages for sending billing and travel logs using both HTTP and HTTPS is shown.

No.	Time	Source	Destination	Proto	Len	Info
16	8.449454	62.44.134.104	172.25.11.18	TCP	74	37655 > x11 [SYN] Seq=0 Win=14600 Len=0 MSS=1380 SACK
17	8.449528	172.25.11.18	62.44.134.104	TCP	74	x11 > 37655 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MS
18	8.479032	62.44.134.104	172.25.11.18	TCP	66	37655 > x11 [ACK] Seq=1 Ack=1 Win=14656 Len=0 TSval=:
19	8.483010	62.44.134.104	172.25.11.18	HTTP	261	POST /RPServer/mobile/billing HTTP/1.1 (text/plain)
20	8.483058	172.25.11.18	62.44.134.104	TCP	66	x11 > 37655 [ACK] Seq=1 Ack=196 Win=30080 Len=0 TSval
21	8.500651	172.25.11.18	62.44.134.104	HTTP	195	HTTP/1.1 200 OK (text/plain)
22	8.533214	62.44.134.104	172.25.11.18	TCP	66	37655 > x11 [ACK] Seq=196 Ack=130 Win=15680 Len=0 TSv

Figure 13.3.1: Billing packages transferred between the mobile client and the server.

A connection is established between the mobile client and the server before the data is sent. In the figure the connection is established in Lines 16 to 18 and uses:

$$74bytes + 74bytes + 66bytes = 208bytes \quad (13.1)$$

In Line 19 the billing is sent to the server. The HTTP request message with the billing uses 261 bytes. An acknowledgment is returned from the server to indicate that it received the package with the billing. This package is shown in Line 20 and uses 66 bytes. The server returns a HTTP response to let the mobile client know that the billing was stored successfully. The HTTP response message uses 195 bytes, which can be seen in Line 21.

Finally, the mobile client sends an acknowledgment to the server to let it know that the response was received. This acknowledgment uses 66 bytes and can be seen in Line 22.

The size of the data sent between the mobile client and the server for each billing is:

$$208bytes + 261bytes + 66bytes + 195bytes + 66bytes = 802bytes \quad (13.2)$$

To perform the data usage calculation we need to use Equation (6.1) from Section 6.1. If a driver drives 60 minutes a day and sends a billing every 15 minutes the following amount of data is used daily:

$$b = \left[ \frac{60min}{15\frac{min}{update}} \right] * 802 \frac{bytes}{update} = 3184bytes \quad (13.3)$$

The data usage for sending billing and travel logs using both HTTP and HTTPS is calculated similarly. The calculations for the data usage of reporting billings, and travel logs with HTTP and HTTPS can be seen in Appendix D.

When calculating the data usage for reporting travel logs we assume that the driver passes 6.47 segments per minute. This is based on actual travel data, which can be seen in Appendix B. In Table 13.3.1 the daily data usage for a driver driving 60 minutes each day and reporting billing and travel logs every 15 minutes can be seen.

	<b>HTTP</b>	<b>HTTPS</b>
Billing	3184 bytes	13040 bytes
Travel Logs	11156 bytes	21068 bytes

Table 13.3.1: Daily data usage of a single driver sending billing and travel logs using HTTP and HTTPS.

We see that reporting a billing costs approximately 3 kB each day using HTTP, and approximately 4 times that using HTTPS. Reporting travel logs costs approximately 11 kB each day using HTTP, and approximately 2 times that using HTTPS.

This illustrates that there is a significant trade-off between security and data usage.

### 13.3.2 Daily Data Usage of Denmark

If we assume that in Denmark 1.5 million drivers drive 60 minutes each day, the daily data usage for the road pricing system in Denmark is:



	<b>HTTP</b>	<b>HTTPS</b>
Billing	4.8 GB	19.6 GB
Travel Logs	16.7 GB	31.6 GB
Total	21.5 GB	51.2 GB

Table 13.3.2: Daily data usage of 1.5 million drivers sending billing and travel logs using HTTP and HTTPS.

### 13.3.3 Internet Connection Requirements

We assume that the data usage peaks during rush hour, and rush hour lasts 4 hours each day, and 40% of the daily traffic is evenly distributed during rush hour. Using these assumptions we can calculate the maximum throughput:

$$\begin{aligned}
\frac{40\% * 51.2GB}{4hours} &= 5.12 \frac{GB}{hour} \\
&\downarrow \\
\frac{5.12 \frac{GB}{hour} * 1000 \frac{MB}{GB}}{3600 \frac{sec}{hour}} &= 1.42 \frac{MB}{s} \\
&\downarrow \\
1.42 \frac{MB}{s} * 8 \frac{Mb}{MB} &= 11.38 \frac{Mb}{s}
\end{aligned} \tag{13.4}$$

Based on the maximum throughput the server will require a flat rate Internet connection of minimum  $11.38 \frac{Mb}{s}$ . This means that we can handle road pricing for Denmark using only a regular household Internet connection. Scaling the road pricing system to a country with more drivers can easily be done by upgrading the Internet connection.

# Chapter 14

## Discussion

In this chapter we will discuss the important aspects of the complete road pricing system.

### 14.1 Field Trial Experiences

In this section we discuss the practical experiences of the field trial.

#### 14.1.1 Price Increase

In the implementation of the system the price displayed in the app is increased when the price of a new segment has been calculated. This means that the size of the increase depends on the taxation model rule associated with the segment and the length of the segment. If a driver encounters a large segment, the cost of driving on the segment will be proportionately large. When this occurs the price will increase significantly as opposed to smoothly, when a driver encounters a new segment frequently.

A test person was surprised of a sudden increase in price because a long segment was encountered.

Usability improvements could be considered in order to display the increase more smoothly. This would, however, not show the precise price, which means there is a trade-off between showing the correct price and usability.

#### 14.1.2 GPS Initialization

When an app that requires a exact GPS location is started on an Android client, it takes some time for the GPS receiver to triangulate the GPS position. We depend on knowing the exact location of the driver in order to charge the correct amount of road pricing tax. This means that while the GPS receiver is triangulating the position, we are unable to perform price calculations. This process may in some cases take considerable time, which meant that some test drivers did not understand why the shown price did not increase.

### 14.1.3 Equipment Failure

Any hardware is prone to failures. An Android client used by a test driver was damaged during testing. This reminds us that at some point any equipment we use will need to be repaired or replaced. It is worth considering the expected lifespan of the onboard unit.

### 14.1.4 Mobile Client Restrictions

A mobile client is restricted in many ways compared to a desktop computer. The processing power, memory, storage capacity, and battery life of the mobile client all need to be considered. During early field trials the map matching algorithm did not perform efficiently enough for the road pricing tax to be displayed in real time. We have optimized the map matching algorithm in order to perform online price calculations.

## 14.2 Accuracy in Map Matching

The accuracy of GPS receivers varies depending on the location. This could be due to GPS satellites being out sight for the receiver e.g. in cities with tall buildings. Therefore map matching is needed in systems that need to associate GPS locations to spatial locations.

As seen in Section 10.3.1 the mobile app implements a simple map matching algorithm. The simple map matching algorithm only uses the distance between the GPS locations and segments as a parameters. This can give problems when segments cross each other or is located side by side.

Map matching algorithms can be partitioned into two groups: online and offline. Online map matching algorithms map match the GPS locations in real-time. Offline map matching algorithms map match a finite set of GPS locations after a trip.

In Table 14.2.1 characteristics of online and offline map matching can be seen. Our simple map matching algorithm lies in the online category.

Online	Offline
<ul style="list-style-type: none"><li>• Gives results in real-time.</li><li>• Low processing time over accuracy.</li><li>• Relies on previous GPS locations.</li></ul>	<ul style="list-style-type: none"><li>• Gives results afterward.</li><li>• Accuracy over low processing time.</li><li>• Can rely on future GPS locations.</li></ul>

Table 14.2.1: Map matching characteristics.

Implementing an offline map matching algorithm is a way to ensure a more accurate result. Offline map matching generally requires more processing power. Even though

processing is limited on a mobile client, performing the map matching on the mobile client is more in line with our goals than performing map matching on the server, as described in Section 3.2. Using an offline map matching algorithm on the mobile client would increase accuracy, but would lower our ability to showcase the road pricing tax. There is a trade-off between accuracy and the ability to showcase the system, which are two of our goals described in Section 2.2.1.

## 14.3 Scalability

In this section the scalability of this system will be discussed. As presented in Section 2.2.1, the system should be able to handle millions of users.

In general scalability methods can be partitioned into two groups.

**Horizontal Scalability** is when the system is shared among multiple servers.

**Vertical Scalability** is when the system is on a single server. This server is then upgraded with many resources.

If maintenance of the system is of high importance vertical scalability would be the way to go. This is because the maintenance would be limited to a single server. On the other hand at some point it will be cheaper to buy several servers than upgrading the single server. Realistically, systems would use a mix between horizontal and vertical scalability.

When it comes to our system a solution could be to use a horizontal scalability approach. This could be various servers that handles a sub-part of the user base. How to partition the users is not a major concern, since the main responsibility of the servers is to store data.

## 14.4 Setup and Operational Costs

In this section we discuss the costs of setting up and operating the road pricing system. We do not calculate exact costs, since too many factors are unknown. We omit discussion on who should pay the different costs.

### 14.4.1 Setup Cost

In order to realize road pricing in the manner we propose, each driver needs to have an Android mobile client. This will generate a setup cost for acquiring the mobile clients. Additionally, servers have to be purchased and set up, or rented through a third party. The setup costs for the system depends on the number of users.

## **14.4.2 Operational Cost**

After the system has been realized operational costs will be associated with it. These costs will be related to the following areas: server, data traffic, user support, and administration.

### **Server**

There can be two kinds of server costs: maintenance and renting. If the server is bought and installed it will need to be maintained for it to function. This may include upgrading the server or replacing parts. There may also be expenses associated with licensing.

If the server is rented a fixed cost will have to be paid regularly. This cost may change if a server is upgraded or downgraded.

### **Data Traffic**

In Section 3.2 the decision to do client-side calculation is made. This means that we reduce the amount of data traffic between the mobile clients and the server. In Section 13.3 we calculate the data traffic between the mobile clients and the server. The calculations are based on the minutes a user drives per day and the frequency at which data is sent. These calculations can help in the calculation of the operational cost. The data traffic price is dependent on the subscriptions for the mobile clients and potential extra costs for roaming.

### **User Support**

We anticipate that with the introduction of a system with a scale like this, there will be users who need support. The cost for supporting users will depend the number of users and the usability of the system.

### **Administration**

For the system to function there is a need for people to administer it. Administration includes the following tasks: creating and maintaining taxation models, keeping the map data up to date, and billing drivers.

## Conclusion

In Chapter 1 we explored other road pricing solutions. Other implementations of road pricing are dependent on custom prototypes.

In Chapter 2 we define our problem statement as:

*How can we analyze, design, and implement all components required for a fully functioning GPS-based road pricing system, such that we are able to make large scale real-world experiments with a complete road pricing system?*

Analyzing the technical options we decided to develop an app for an Android client. This was in line with our scalability goal because Android has the largest market share of smartphones, which means many people are able to run our mobile app on their own device. Additionally, an Android client satisfied all our requirements for an onboard unit.

We chose to implement a client/server architecture with four databases. Having multiple databases is used to separate privacy-sensitive data from other data. This also allows for having different parties handle different types of data. The location of the drivers is stored in the travel log database, while the billing information is stored in the billing database.

We created a web application that showcases travel and billing information to drivers, road pricing providers, and audit users, while handling the permissions of the user types.

We create a novel message format that has significantly lower message size than popular formats such as JSON and XML. This lowers the overall data usage and thereby also the requirements and pressure of the network. The established message formats were 80% to 290% greater in size than our novel message format, as described in Section 13.2.

Using the novel message format we calculated the required Internet connection to support road pricing in Denmark to be only 11.38Mb/s during rush hour, which is nothing more than a regular household Internet connection.

To ensure privacy all data transfers are encrypted by the use of HTTPS.

Through testing we found that the map matching algorithm finds close to all correct

segments (93.9% to 98%), but approximately (22.6% to 23.8%) a quarter of the segments found were not part of the trip.

We implemented all required components for a fully-functional complete road pricing system and performed extensive field trials. The field trials were performed over a 8 week period and included 7 test drivers, who drove over 4000 kilometers combined, and collected over 10000 travel logs and over 800 billings.

The functional road pricing system fulfilled our goals described in Section 2.2.1:

- We are able to showcase the functional system in real-time through the Android app we developed, and in-depth travel and billing information can be accessed through the web application we developed. This was successfully tested through field trials.
- The map matching is accurate enough to find nearly all segments of a trip, albeit with some additional segments.
- We ensure privacy by encrypting all traffic and separating privacy-sensitive data.
- We keep costs low by performing price calculations client-side and limiting data traffic by using a novel message format.
- We ensure that the system is scalable by performing the heavy calculations client-side, so the pressure on the servers is low. Additionally, the mobile client we use is an Android device, which has the largest market share of smartphones therefore supporting a large user base.

This means that we have successfully created all necessary components for a fully functioning complete GPS-based road pricing system that allows for large scale real-world experiments.

# Chapter 16

## Future Work

In this chapter we describe the parts of the project we would continue to work on if we had more time.

### 16.1 Map Matching

Map matching is a complex research topic with many competing algorithms. Implementing map matching is a large task, and finding a readily usable algorithm presents some challenges: The processing power on our mobile client is limited, which means that a map matching algorithm has to be very efficient not to become a bottleneck. Determining the accuracy of a map matching algorithm is difficult without performing actual experiments with an implemented algorithm. This is a time-consuming task.

Time should either be spent on implementing a more accurate map matching algorithm, or alternatively we could purchase a commercial map matching solution.

### 16.2 Data Encryption

At the moment the *TravelLog* information is stored without encryption in the travel log database. This is not in line with our privacy goal in Section 2.2.1. The travel log data is stored in a separate database, which makes it possible to store it on a separate server.

The data in the travel log database should be encrypted, such that the party hosting the travel log database is unable to view sensitive data. This has the added effect that if information is leaked, or if someone gains access to the database, private data cannot be read.



## 16.3 User Management

User information is typically very privacy-sensitive, which means that the implementation for handling user data should be very secure. The user information is stored in a separate database. This allows for a third party to handle user information.

In Denmark the ideal solution would be to use the already implemented NemID solution [11] to handle user information and logins. By using existing solutions like NemID, users would only need to remember one login. Furthermore, the security would be outsourced to companies that are experts on the area.

## 16.4 Message Reduction

Even though the byte size of our novel message format has already been greatly reduced, it is possible to reduce them even further. This is in line with our scalability and low operational costs goals, as described in Section 2.2.1

In Section 12.3.1 the implementation of the TravelLog Message Format was introduced. These can be fine-tuned to take up a couple of less bytes, but we can get a great gain by using a variant of delta encoding.

In this variant the mobile client and the server would have a copy of a standard message. Then instead of sending a whole message the mobile client would send the difference between the message and the standard message. The server will then compare the received message with the standard message to translate it into the original message. This would for instance eliminate the need for sending the *Taxation Model ID* with every message.

## 16.5 Fraud Prevention

There will always be users that will try to exploit the system, which is why we have to prevent that.

Users could spoof their location, which means that they would be billed for driving somewhere that they are not. To prevent users that try to spoof their GPS location, the mobile app could check if *mock locations* is enabled on the mobile client. This will only stop the users that use a mobile client, which is not rooted. Rooted means mobile clients that can access system files.

Furthermore, the mobile app could try to see if the distance between two consecutive GPS locations is within a realistic range.

At last the user could try to drive with the mobile app having the GPS receiver disabled. This can be prevented by having some kind of “heart beat”, which checks once in a while if everything is functioning correctly.

These modifications would inhibit GPS spoofing, but also increase the processing requirements.

# Bibliography

- [1] 3GPP. 3rd generation partnership project; technical specification group core network; numbering, addressing and identification. [http://www.arib.or.jp/english/html/overview/doc/STD-T63v9\\_30/5\\_Appendix/Rel5/23/23003-5b0.pdf](http://www.arib.or.jp/english/html/overview/doc/STD-T63v9_30/5_Appendix/Rel5/23/23003-5b0.pdf), 2006. Last Viewed: 2014.05.22.
- [2] Tommi Aihkisalo and Tuomas Paaso. A performance comparison of web service object marshalling and unmarshalling solutions. *IEEE*, 2011. Last Viewed: 2014.02.24.
- [3] Tommi Aihkisalo and Tuomas Paaso. Latencies of service invocation and processing of the rest and soap web service interfaces. *IEEE*, 2012. Last Viewed: 2014.02.24.
- [4] Strategy Analytics. Android captures record 81 percent share of global smartphone shipments in q3 2013. <http://blogs.strategyanalytics.com/WSS/post/2013/10/31/Android-Captures-Record-81-Percent-Share-of-Global-Smartphone-Shipments-in-Q3-2013.aspx>, October 2013. Last Viewed: 2014.02.21.
- [5] Strategy Analytics. Android pushes past 80leap 156.0 <http://www.idc.com/getdoc.jsp?containerId=prUS24442013>, November 2013. Last Viewed: 2014.02.21.
- [6] Apache. Apache tomcat. <http://tomcat.apache.org/>, 2014. Last Viewed: 2014.03.03.
- [7] John Blau. High-tech truck toll system finally launched in germany. [http://www.computerworld.com/s/article/98679/High\\_tech\\_truck\\_toll\\_system\\_finally\\_launched\\_in\\_Germany](http://www.computerworld.com/s/article/98679/High_tech_truck_toll_system_finally_launched_in_Germany), 2005. Last Viewed: 2014.05.29.
- [8] Laura Blow, Andrew Leicester, and Zoë Smith. London's congestion charge. <http://eprints.ucl.ac.uk/14932/1/14932.pdf>, 2003. Last Viewed: 2014.05.29.
- [9] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen Satish Thatte, and Dave Winer. Simple object access protocol (soap) 1.1, May 2000. URL <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.

- [10] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible markup language (xml) 1.0 (fifth edition), November 2008. URL <http://www.w3.org/TR/2008/REC-xml-20081126/>.
- [11] Nets DanID. Nemid. <https://www.nemid.nu/dk-da/>, 2010. Last Viewed: 2014.05.30.
- [12] Inc. E. Rescorla RTFM. Http over tls. <https://tools.ietf.org/html/rfc2818>, May 2000. Last Viewed: 2014.05.05.
- [13] Eclipse. Jetty. <http://www.eclipse.org/jetty/>, 2014. Last Viewed: 2014.03.03.
- [14] Roy Thomas Fielding. Representational state transfer (rest). [http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm), 2000. Last Viewed: 2014.02.24.
- [15] International Organization for Standardization (ISO). Information processing - documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts. <https://www.iso.org/obp/ui/#iso:std:iso:5807:ed-1:v1:en>, 1985. Last Viewed: 2014.05.27.
- [16] Department for Transport. Road pricing demonstrations project - introduction. <https://www.gov.uk/government/publications/road-pricing-demonstrations-project-introduction>, 2011. Last Viewed: 2014.05.29.
- [17] Department for Transport. Wp07 design of in-vehicle equipment to support road pricing. <http://lists.umn.edu/cgi-bin/wa?A3=ind1105&L=CON-PRIC&E=base64&P=10237118&B=-005045016d7d4a3cff04a41cca3b&T=application%2Fpdf;%20name=%22wp07%20invehicledesign.pdf%22&N=wp07%20invehicledesign.pdf&attachment=q>, 2011. Last Viewed: 2014.05.29.
- [18] Alessandro Furieri. Spatialite on android: a quick tutorial. <https://www.gaia-gis.it/fossil/libspatialite/wiki?name=spatialite-android-tutorial>, 2013. Last Viewed: 2014.02.24.
- [19] AGES ETS GmbH. Eurovignette. <https://www.eurovignettes.eu/portal/>, 2014. Last Viewed: 2014.05.29.
- [20] Google. Protocol buffers, 2012. URL <https://developers.google.com/protocol-buffers/>.
- [21] PHP Group. Php: Hypertext preprocessor. <http://www.php.net/>, February 2014. Last Viewed: 2014.02.24.
- [22] GSMARENA. Lg nexus 5. [http://www.gsmarena.com/lg\\_nexus\\_5-5705.php](http://www.gsmarena.com/lg_nexus_5-5705.php), 2013. Last Viewed: 2014.05.22.
- [23] Red Hat. Jboss application server. <http://www.jboss.org/overview/>, 2013. Last Viewed: 2014.01.09.

- [24] Apple Inc. Mac app store – xcode. <https://itunes.apple.com/us/app/xcode/id497799835?ls=1&mt=12>, 2014. Last Viewed: 2014.02.21.
- [25] Google Inc. Activities. <http://developer.android.com/guide/components/activities.html>, 2014. Last Viewed: 2014.05.27.
- [26] Google Inc. Ice cream sandwich. <http://developer.android.com/about/versions/android-4.0-highlights.html>, February 2014. Last Viewed: 2014.02.21.
- [27] Google Inc. Legal notice — android developers. <http://developer.android.com/legal.html>, 2014. Last Viewed: 2014.02.21.
- [28] Google Inc. Storage options — android developers. <http://developer.android.com/guide/topics/data/data-storage.html>, February 2014. Last Viewed: 2014.02.21.
- [29] Google Inc. Services. <http://developer.android.com/guide/components/services.html>, 2014. Last Viewed: 2014.05.27.
- [30] T. Dierks Independent and Inc. E. Rescorla RTFM. The transport layer security (tls) protocol version 1.2. <https://tools.ietf.org/html/rfc5246>, August 2008. Last Viewed: 2014.05.05.
- [31] Ecma International. The json data interchange format, October 2013. URL <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>. Last Viewed: 2014.02.24.
- [32] R. Fielding UC Irvine, J. Gettys Compaq/W3C, J. Mogul Compaq, H. Frystyk W3C/MIT, L. Masinter Xerox, P. Leach Microsoft, and T. Berners-Lee W3C/MIT. Hypertext transfer protocol – http/1.1. <https://tools.ietf.org/html/rfc2616>, Juni 1999. Last Viewed: 2014.05.05.
- [33] Usman Ismail. A case against using protobuf for transport in rest services. <http://techtraits.com/noproto/>, April 2013. Last Viewed: 2014.02.24.
- [34] Solid IT. Db-engines ranking. <http://db-engines.com/en/ranking>, 2014. Last Viewed: 2014.03.03.
- [35] Jelastic. Java & php software stacks market share: October 2013. [http://blog.jelastic.com/2013/11/07/software-stacks-market-share-october-2013/?utm\\_source=tuicool](http://blog.jelastic.com/2013/11/07/software-stacks-market-share-october-2013/?utm_source=tuicool), 2013. Last Viewed: 2014.03.10.
- [36] Øresundsbro Konsortiet. The oresund bridge. <http://uk.oresundsbron.com/page/976>, 2014. Last Viewed: 2014.05.29.
- [37] Jean loup Gailly. The gzip home page. <http://www.gzip.org/>, July 2003. Last Viewed: 2014.03.07.

- [38] Simon Maple. The great java application server debate with tomcat, jboss, glassfish, jetty and liberty profile. <http://zeroturnaround.com/rebellabs/the-great-java-application-server-debate-with-tomcat-jboss-glassfish-jetty-and-liberty-profile/>, Maj 2013. Last Viewed: 2014.03.04.
- [39] Microsoft. The official microsoft asp.net site. <http://www.asp.net/>, 2014. Last Viewed: 2014.03.10.
- [40] Microsoft. Microsoft sql server. <http://www.microsoft.com/en-us/sqlserver/default.aspx>, 2014. Last Viewed: 2014.03.03.
- [41] Dan Stenholt Møller, Jens Mohr Mortensen, and Mikael Midtgaard. Road pricing calculation system. Master's thesis, Aalborg University, 2013. Last Viewed: 2014.05.29.
- [42] Jens Mohr Mortensen. J1stracking. <http://j1s.dk/tracking/?date=20140601&clientid=j1s>, 2014. Last Viewed: 2014.06.01.
- [43] Oracle. Glassfish. <https://glassfish.java.net/>, 2013. Last Viewed: 2014.03.03.
- [44] Oracle. Javasever faces technology. <http://www.oracle.com/technetwork/java/javase/javasefaces-139869.html>, 2013. Last Viewed: 2014.01.09.
- [45] Oracle. Oracle database. <http://www.oracle.com/us/products/database/overview/index.html>, 2014. Last Viewed: 2014.03.03.
- [46] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. "big" web services: Making the right architectural decision. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 805–814, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-085-2. doi: 10.1145/1367497.1367606. URL <http://doi.acm.org/10.1145/1367497.1367606>. Last Viewed: 2014.02.24.
- [47] PostGIS. Documentation. <http://postgis.net/documentation>, 2013. Last Viewed: 2013.12.09.
- [48] PostgreSQL. Postgresql. <http://www.postgresql.org/>, 2013. Last Viewed: 2013.12.09.
- [49] roadtraffic technology.com. Lkw-maut electronic toll collection system for, germany. <http://www.roadtraffic-technology.com/projects/lkw-maut/>, 2006. Last Viewed: 2014.05.29.
- [50] SQLite. Datatypes in sqlite version 3. <http://www.sqlite.org/datatype3.html>, 2014. Last Viewed: 2014.03.06.
- [51] SQLite. Appropriate uses for sqlite. <http://www.sqlite.org/whentouse.html>, 2014. Last Viewed: 2014.03.06.

- [52] A/S Storebælt. Storebælt. <http://www.storebaelt.dk/>, 2014. Last Viewed: 2014.05.29.
- [53] [www.dalnicni-znamky.com](http://www.dalnicni-znamky.com). Vignettes and highway toll in europe. <http://www.dalnicni-znamky.com/en/>, 2014. Last Viewed: 2014.05.29.
- [54] Martina Zanic. *GNSS-based Road Charging Systems Assessment of Vehicle Location Determination*. PhD thesis, Technical University of Denmark, 2011. Last Viewed: 2014.05.29.



**Part V**

**Appendices**





# Taxation Model Database Schema

In Figure A.0.1 the database schema from RCS [41, p. 31] can be seen. This database schema defines a taxation model as a number of areas. Any combination of a taxation model and an area can be populated with rules and prices for these. A rule applies to a segment, road type, or border crossing in a given time interval.

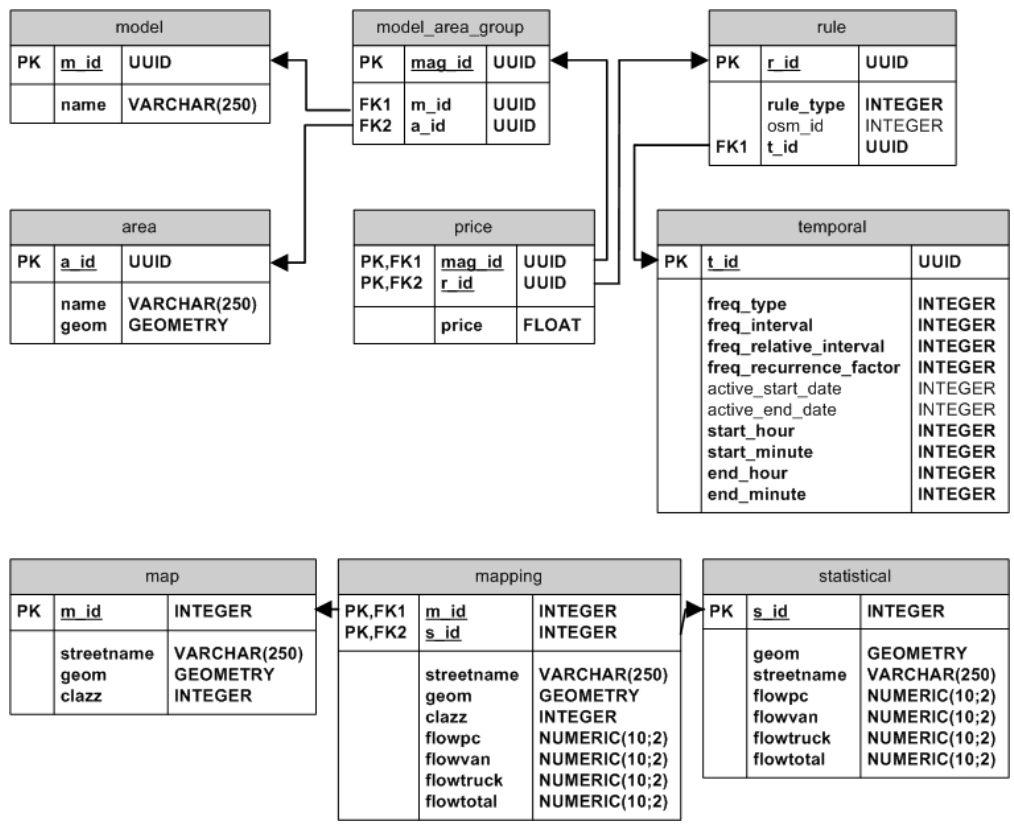


Figure A.0.1: This is the database schema for the taxation model of RCS.



# Appendix B

## Travel Data

Table B.0.1 shows travel data generated by using the Dijkstra shortest path between cities on the map data of the system. The letter in parenthesis describes the types of road of the route. M represents motorway, H represents highway, and C represent city.

Route	Segments	Duration	Segments/Minute
Aalborg - Randers (M)	96	50 min	1.92
Aalborg - Blokhus (MH)	106	40 min	2.65
Holstebro - Randers (H)	282	90 min	3.13
Holstebro - Esbjerg (H)	264	88 min	3
Aarhus - Randers (M)	105	33 min	3.18
Grenaa - Randers (H)	186	55 min	3.38
Odense - Esbjerg (M)	165	83 min	1.99
Klampenborg - Christianshavn (C)	107	21 min	5.10
Vanløse - Christianshavn (C)	97	15 min	6.47
Klampenborg - Christianshavn (C)	41	8 min	5.13

Table B.0.1: Something

	segment/minute	minute/segment
Min	1.92	0.52
Avg	3.59	0.28
Max	6.47	0.15

Table B.0.2: Something



# Appendix C

## Data Usage

In Figure C.0.1 the data packages sent when reporting a billing with HTTP can be seen.

No.	Time	Source	Destination	Proto	Len	Info
16	8.449454	62.44.134.104	172.25.11.18	TCP	74	37655 > x11 [SYN] Seq=0 Win=14600 Len=0 MSS=1380 SACK_PERM=1 TSval=69676564 TSecr=69676564 Win=0
17	8.449528	172.25.11.18	62.44.134.104	TCP	74	x11 > 37655 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=69676564 TSecr=69676564 Win=0
18	8.479032	62.44.134.104	172.25.11.18	TCP	66	37655 > x11 [ACK] Seq=1 Ack=1 Win=14656 Len=0 TSval=69676564 TSecr=69676564 Win=0
19	8.483010	62.44.134.104	172.25.11.18	HTTP	261	POST /RPServer/mobile/billing HTTP/1.1 (text/plain)
20	8.483058	172.25.11.18	62.44.134.104	TCP	66	x11 > 37655 [ACK] Seq=1 Ack=196 Win=30080 Len=0 TSval=69676564 TSecr=69676564 Win=0
21	8.500651	172.25.11.18	62.44.134.104	HTTP	195	HTTP/1.1 200 OK (text/plain)
22	8.533214	62.44.134.104	172.25.11.18	TCP	66	37655 > x11 [ACK] Seq=196 Ack=130 Win=15680 Len=0 TSval=69676564 TSecr=69676564 Win=0

Figure C.0.1: Billing packages transferred between the mobile client and the server.

In Figure C.0.2 the data packages sent when reporting a billing with HTTPS can be seen.

No.	Time	Source	Destination	Proto	Len	Info
11	8.725643	62.44.134.26	172.25.11.18	TCP	74	51175 > x11 [SYN] Seq=0 Win=14600 Len=0 MSS=1380 SACK_PERM=1 TSval=46243 TSecr=69676564 Win=0
12	8.725717	172.25.11.18	62.44.134.26	TCP	74	x11 > 51175 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=69676564 TSecr=69676564 Win=0
13	8.752983	62.44.134.26	172.25.11.18	TCP	66	51175 > x11 [ACK] Seq=1 Ack=1 Win=14656 Len=0 TSval=46243 TSecr=69676564 Win=0
14	8.756821	62.44.134.26	172.25.11.18	TCP	250	51175 > x11 [PSH, ACK] Seq=1 Ack=1 Win=14656 Len=184 TSval=46243 TSecr=69676564 Win=0
15	8.756865	172.25.11.18	62.44.134.26	TCP	66	x11 > 51175 [ACK] Seq=1 Ack=185 Win=30080 Len=0 TSval=69676572 TSecr=46243 Win=0
16	8.761903	172.25.11.18	62.44.134.26	TCP	1225	x11 > 51175 [PSH, ACK] Seq=1 Ack=185 Win=30080 Len=1159 TSval=69676573 TSecr=46243 Win=0
17	8.800907	62.44.134.26	172.25.11.18	TCP	66	51175 > x11 [ACK] Seq=185 Ack=1160 Win=17536 Len=0 TSval=46247 TSecr=69676573 Win=0
18	8.808557	62.44.134.26	172.25.11.18	TCP	376	51175 > x11 [PSH, ACK] Seq=185 Ack=1160 Win=17536 Len=310 TSval=46247 TSecr=69676573 Win=0
19	8.848429	172.25.11.18	62.44.134.26	TCP	66	x11 > 51175 [ACK] Seq=1160 Ack=495 Win=31104 Len=0 TSval=69676595 TSecr=46247 Win=0
20	8.854154	172.25.11.18	62.44.134.26	TCP	72	x11 > 51175 [PSH, ACK] Seq=1160 Ack=495 Win=31104 Len=6 TSval=69676596 TSecr=46247 Win=0
21	8.854812	172.25.11.18	62.44.134.26	TCP	103	x11 > 51175 [PSH, ACK] Seq=1166 Ack=495 Win=31104 Len=37 TSval=69676596 TSecr=46247 Win=0
22	8.888712	62.44.134.26	172.25.11.18	TCP	66	51175 > x11 [ACK] Seq=495 Ack=1203 Win=17536 Len=0 TSval=46256 TSecr=69676596 Win=0
23	8.896712	62.44.134.26	172.25.11.18	TCP	282	51175 > x11 [PSH, ACK] Seq=495 Ack=1203 Win=17536 Len=216 TSval=46257 TSecr=69676596 Win=0
24	8.896750	172.25.11.18	62.44.134.26	TCP	66	x11 > 51175 [ACK] Seq=1203 Ack=711 Win=32256 Len=0 TSval=69676607 TSecr=46257 Win=0
25	8.902519	172.25.11.18	172.25.18.16	SSH	242	Encrypted response packet len=176
26	8.902786	172.25.11.18	172.25.18.16	SSH	114	Encrypted response packet len=48
27	8.904522	172.25.18.16	172.25.11.18	TCP	66	53536 > ssh [ACK] Seq=1 Ack=177 Win=1444 Len=0 TSval=8646746 TSecr=69676608 Win=0
28	8.904585	172.25.18.16	172.25.11.18	TCP	66	53536 > ssh [ACK] Seq=1 Ack=225 Win=1444 Len=0 TSval=8646746 TSecr=69676608 Win=0
29	8.915237	172.25.11.18	62.44.134.26	TCP	216	x11 > 51175 [PSH, ACK] Seq=1203 Ack=711 Win=32256 Len=150 TSval=69676611 TSecr=46257 Win=0
30	8.981240	62.44.134.26	172.25.11.18	TCP	66	51175 > x11 [ACK] Seq=711 Ack=1353 Win=19840 Len=0 TSval=46266 TSecr=69676611 Win=0

Figure C.0.2: HTTPS billing packages transferred between the mobile client and the server.

In Figure C.0.3 the data packages sent when reporting travel logs with HTTP can be seen.

No.	Time	Source	Destination	Proto	Len	Info
33	30.166938	172.25.21.230	172.25.11.18	TCP	74	60722 > x11 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK
34	30.166997	172.25.11.18	172.25.21.230	TCP	74	x11 > 60722 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=
35	30.171936	172.25.21.230	172.25.11.18	TCP	66	60722 > x11 [ACK] Seq=1 Ack=1 Win=14656 Len=0 TSval=68
36	30.186352	172.25.21.230	172.25.11.18	HTTP	268	POST /RPServer/mobile/travellog HTTP/1.1 (text/plain)
37	30.186399	172.25.11.18	172.25.21.230	TCP	66	x11 > 60722 [ACK] Seq=1 Ack=203 Win=30080 Len=0 TSval=f
38	30.213227	172.25.11.18	172.25.21.230	HTTP	195	HTTP/1.1 200 OK (text/plain)
39	30.215864	172.25.21.230	172.25.11.18	TCP	66	60722 > x11 [ACK] Seq=203 Ack=130 Win=15680 Len=0 TSva

Figure C.0.3: Travel log packages transferred between the mobile client and the server.

In Figure C.0.4 the data packages sent when reporting travel logs with HTTPS can be seen.

No.	Time	Source	Destination	Proto	Len	Info
127	24.932148	172.25.18.127	172.25.11.18	TCP	74	47074 > x11 [SYN] Seq=0 Win=12400 Len=0 MSS=1240 SACK_PERM=1 TSval=52196317 TSecr=0 WS=6
128	24.932261	172.25.11.18	172.25.18.127	TCP	74	x11 > 47074 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=957988054
129	24.933873	172.25.18.127	172.25.11.18	TCP	66	47074 > x11 [ACK] Seq=1 Ack=1 Win=12416 Len=0 TSval=52196317 TSecr=957988054
130	24.944989	172.25.18.127	172.25.11.18	TCP	258	47074 > x11 [PSH, ACK] Seq=1 Ack=1 Win=12416 Len=184 TSval=52196319 TSecr=957988054
131	24.945037	172.25.11.18	172.25.18.127	TCP	66	x11 > 47074 [ACK] Seq=1 Ack=185 Win=30080 Len=0 TSval=957988057 TSecr=52196319
132	24.947045	172.25.11.18	172.25.18.127	TCP	1225	x11 > 47074 [PSH, ACK] Seq=1 Ack=185 Win=30080 Len=1159 TSval=957988057 TSecr=52196319
133	24.949069	172.25.18.127	172.25.11.18	TCP	66	47074 > x11 [ACK] Seq=185 Ack=1160 Win=14720 Len=0 TSval=52196320 TSecr=957988057
134	24.970116	172.25.18.127	172.25.11.18	TCP	376	47074 > x11 [PSH, ACK] Seq=185 Ack=1160 Win=14720 Len=310 TSval=52196324 TSecr=957988057
135	25.007406	172.25.11.18	172.25.18.127	TCP	66	x11 > 47074 [ACK] Seq=1160 Ack=495 Win=31104 Len=0 TSval=957988073 TSecr=52196324
136	25.031406	172.25.11.18	172.25.18.127	TCP	72	x11 > 47074 [PSH, ACK] Seq=1160 Ack=495 Win=31104 Len=6 TSval=957988079 TSecr=52196324
137	25.032171	172.25.11.18	172.25.18.127	TCP	103	x11 > 47074 [PSH, ACK] Seq=1160 Ack=495 Win=31104 Len=37 TSval=957988079 TSecr=52196324
138	25.036227	172.25.18.127	172.25.11.18	TCP	66	47074 > x11 [ACK] Seq=495 Ack=1203 Win=14720 Len=0 TSval=52196338 TSecr=957988079
139	25.067423	172.25.18.127	172.25.11.18	TCP	309	47074 > x11 [PSH, ACK] Seq=495 Ack=1203 Win=14720 Len=243 TSval=52196344 TSecr=957988079
140	25.067468	172.25.11.18	172.25.18.127	TCP	66	x11 > 47074 [ACK] Seq=1203 Ack=738 Win=32256 Len=0 TSval=957988088 TSecr=52196344
141	25.115868	172.25.11.18	172.25.18.127	TCP	216	x11 > 47074 [PSH, ACK] Seq=1203 Ack=738 Win=32256 Len=150 TSval=957988100 TSecr=52196344
142	25.155742	172.25.18.127	172.25.11.18	TCP	66	47074 > x11 [ACK] Seq=738 Ack=1353 Win=17088 Len=0 TSval=52196362 TSecr=957988100

Figure C.0.4: HTTPS travel log packages transferred between the mobile client and the server.

# Appendix D

## Data Usage Calculations

Equation (D.1) shows the data usage calculation for reporting billing using HTTP for a driving driving 60 minutes per day.

$$b = \left[ \frac{60min}{15 \frac{min}{update}} \right] * 802 \frac{bytes}{update} = 3184bytes \quad (D.1)$$

Equation (D.2) shows the data usage calculation for reporting billing using HTTPS for a driving driving 60 minutes per day.

Equation (D.2)

$$b = \left[ \frac{60min}{15 \frac{min}{update}} \right] * 3260 \frac{bytes}{update} = 13040bytes \quad (D.2)$$

Equation (D.2) shows the data usage calculation for reporting travel logs using HTTP for a driving driving 60 minutes per day. The driver encounters a new segment once every 0.15 minutes.

Equation (D.3)

$$l = \left[ \frac{60min}{15 \frac{min}{update}} \right] * (745 + 44) \frac{bytes}{update} + \left[ \frac{60min}{0.15 \frac{min}{segment}} \right] * 20 \frac{bytes}{segment} = 11156bytes \quad (D.3)$$

Equation (D.2) shows the data usage calculation for reporting travel logs using HTTPS for a driving driving 60 minutes per day. The driver encounters a new segment once every 0.15 minutes.

Equation (D.4)

$$l = \left[ \frac{60min}{15 \frac{min}{update}} \right] * (3223 + 44) \frac{bytes}{update} + \left[ \frac{60min}{0.15 \frac{min}{segment}} \right] * 20 \frac{bytes}{segment} = 21068bytes \quad (D.4)$$





Appendix **E**

**CD**