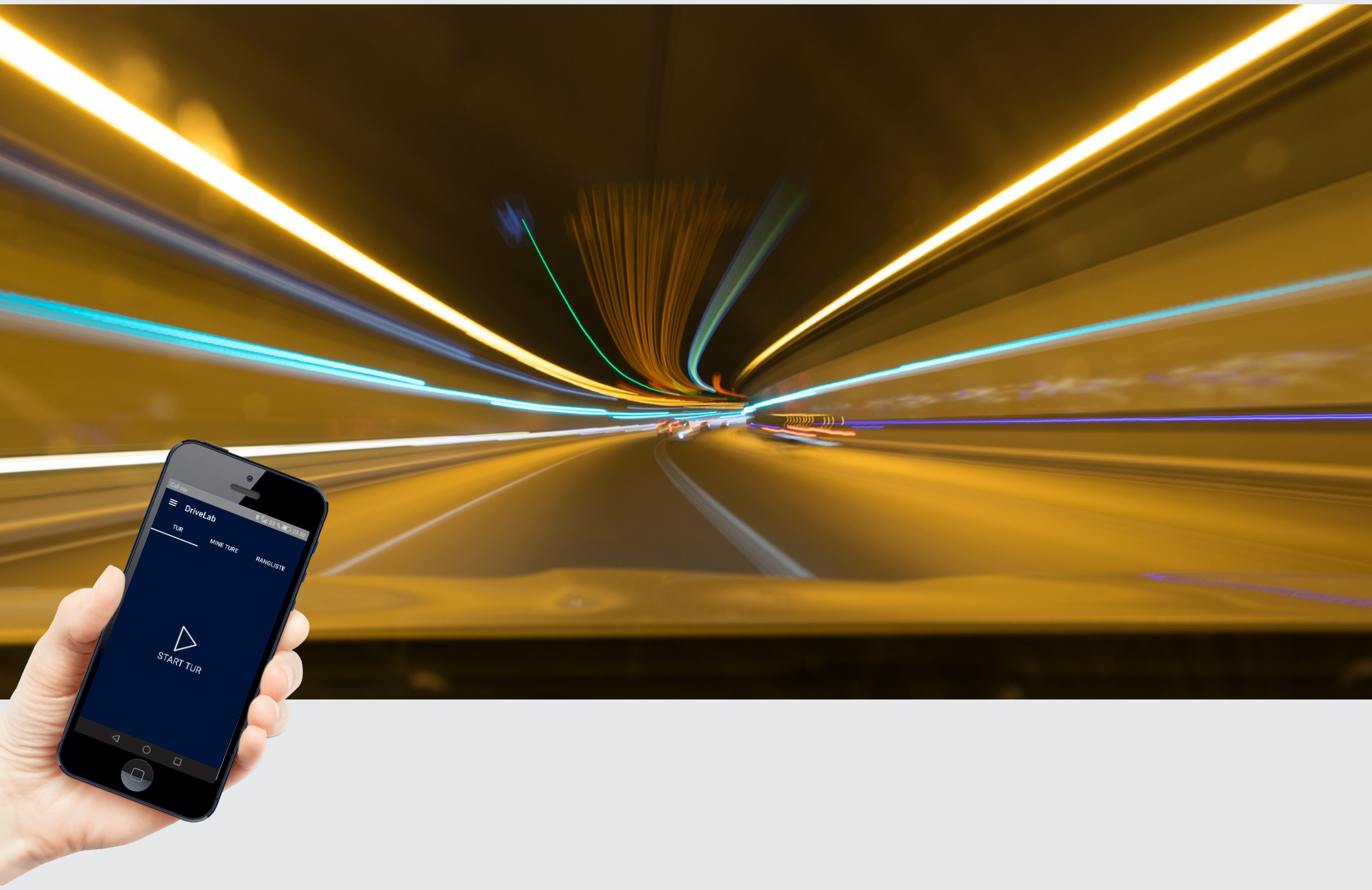
AALBORG UNIVERSITY
STUDENT REPORT



DriveLaB: A Speeding Reductive Mobile Crowd Sensing Platform

Dennis Rasmussen, Kasper F. Pedersen, Thomas F. Olsen

Master Thesis

P10 Master Project
DriveLaB
MSc. Computer Science (IT)
Aalborg University
2017

AALBORG UNIVERSITY
STUDENT REPORT

Title	DriveLaB
Semester:	10th semester at MSc. Computer Science (IT)
Semester theme:	DriveLaB
Project period:	01/02/17 to 09/06/17
ECTS:	30
Supervisor:	Kristian Torp
Project group:	Group dpt107f17

Abstract:

In this report, a speeding reductive *Mobile Crowd Sensing* platform is designed, implemented and evaluated. The platform includes a client application available for Android and iOS with speeding reductive measures incorporated. The server performs real-time map matching, data calculations, and provides real-time feedback. The data is stored in a data warehouse which enables the platform to offer a public API with anonymised data. Users are encouraged to participate through the client application by offering traffic related feedback, calculate driving scores and displaying a competitive leaderboard.

Dennis Rasmussen

Kasper Fromm Pedersen

Thomas Frisk Olsen

Pages:	75 pages
Appendix:	9 pages

By signing this document, each member of the group confirms participation on equal terms in the process of writing the project. Thus, each member of the group is responsible for all contents in the project.

Summary

Road traffic accidents carry significant economic consequences for the society and grief for relatives in case of severe accidents. Speeding is the main contributor to the cause of accidents, as it is solely or partly responsible for 41% of all accidents. The Danish Road Safety Commission has set ambitious goals of reducing road traffic accidents and identifies technology as an enabler to achieve these goals.

In this report, a speeding reductive *Mobile Crowd Sensing* platform is designed, implemented and evaluated. The platform includes a client application available for Android and iOS with speeding reductive measures incorporated, which communicates in real-time with a server. The server performs real-time map matching, data calculations, and provides real-time feedback. The data is stored in a data warehouse which enables the platform to offer a public API. This API puts anonymised data collected by the client application at disposal for everyone to use. Users are encouraged to participate through the client application by offering traffic related feedback, calculate driving scores and displaying a competitive leaderboard.

The platform seeks to solve a contextual issue regarding missing knowledge of the Danish road speed limits in *Open Street Map*. The solution is to implement a heuristic algorithm to deduce speed limits from OSM road classifications. Additionally, users can report speed limits using speech recognition, which are processed by a reputation system.

The platform is evaluated through controlled and uncontrolled experiments to investigate system stability, system performance, behaviour change in users, an automatic tracking solution, and scoring fairness.

Preface

This report extends a series of student publications [1][2][3] conducted at Department of Computer Science, Aalborg University. However, only [3] is composed by the same authors as this report. As in the previous projects, a collaboration with a Danish insurance company Lærerstandens Brandforsikring (LB) benefits the research in respect to data collection and business insight.

The previous report [3] aimed to develop a full stack software system that should be able to monitor road users in real-time as well as provide driving-related feedback in a safely manner. The system consisted of three important components.

- A smartphone client application, targeting the Android operating system using Ionic 2, a web-based framework for developing cross-platform smartphone applications.
- A middle-end server application, using ASP.NET Core framework, making the server side software cross platform.
- A back-end server, running the freely available PostgreSQL database, to avoid licensing concerns.

Additionally, it was deemed necessary to collect GPS information without requiring the user to interact with the system. To solve this, two devices at two different price levels were introduced. The expensive one, a Garmin GLO [4], which provides a 10 Hz external GPS receiver accessible over a Bluetooth classic connection. The cheapest one, a Kontakt.io beacon [5] which supports both the iBeacon and Eddystone protocol. To ensure continuous system use, incentives to keep the drivers using the smartphone application was also considered. Among these incentives, a scoring system solely based on the speed provided by the GPS receiver was suggested to render a competitive leaderboard possible. Finally, an experiment was conducted in collaboration with LB, involving 23 participants, covering 6455 km of road in total.

The report concludes that GPS receivers provided by modern smartphones are uniform enough for the scoring algorithm presented.

Future work presented in [3] will be the underlying basis for this report. This includes improvements, architectural changes, and feature additions to the full stack software system previously presented. Furthermore, adjustments will be made to incorporate a crowd sourced reporting system for digital speed limits, as the partial lack of these has shown to influence the user experience negatively. The smartphone application presented in previous work [3], and worked exclusively for Android. This constrained the extent of which LB could offer participation in the experiment as iPhone is the most representative phone in that company. Therefore, an iPhone application will also be developed, even though the scope of cross platform development is not relevant for the overall project subject.

LB have decided to remain in the field of facilitating Corporate Social Responsibility

(CSR). Therefore, this is still considered an important aspect of the full stack software solution presented.

Reading Guide

This report consists of six chapters. The first chapter, *Introduction*, describes the problem attempted to be solved and the context of this project.

Chapter two, *Related work*, presents previous work which this project uses as basis or inspiration to solve the challenges at hand.

Chapter three, *Analysis & Design*, describes the five-layer model and its use in this project. The model consists of the following five components: *Mobile Crowd Sensing*, *Data Transmission*, *Data Collection and Real-time Processing*, *Crowd Data Processing*, and *Application*, which each have their purpose in the system.

Chapter four, *Implementation*, presents implementation details for each of the components in the system.

Chapter five, *Evaluation*, subjects the system to evaluation through both controlled and uncontrolled experiments. Data analysis is performed on the data gathered throughout the experiments performed.

Chapter six, *Reflections*, discusses the findings during this project. Furthermore, a conclusion will be presented alongside suggestions for further work.

Contents

Introduction 1

In 2015, 11.105 traffic accidents were registered on Danish roads. 3.334 of these accidents included person injury, and 178 of these had a fatal exit [6]. Each of the 11.105 accidents cost 600.000-700.000 DKK on average [7], as few accidents conflicting severe personal injury drives up the costs. One example of such, created by the National Road Directorate, is based on a real accident where the treatment of one person's costs exceeded 11 million DKK for the first five years after the accident, followed by 2.2 million per year for the rest of the injured person's life [8]. In 2012, the average society economic cost, per person injury, was calculated to be 4,6 million DKK. This cover treating the person injuries, the cost of material damage and loss in welfare by the person not being able to contribute to society after the accident.

The Danish Road Safety Commission (RSC) works to improve the road safety in Denmark by formulating a plan of action and setting goals for road safety. The plan of action laid down for the years 2001 to 2012 accomplished the traffic safety goals, and the number of injured and killed in traffic was halved. The amount of injured and killed people in traffic is in fact at an all-time low since 1930, where accident registering started [9]. However, working under the vision "*Every Accident is one too many – a shared responsibility*", the road safety goals for 2013 to 2020 is even more ambitious. The goals are set for a maximum of 120 killed, 1000 severely injured and 1000 minor injured in traffic by 2020 [9].

The severity of traffic accidents makes the area of traffic safety an attractive opportunity for companies to display Corporate Social Responsibility (CSR) within. A well-known example hereof is the Danish company Tryg, which includes the Tryg foundation. This foundation explicitly has fewer injuries and casualties in traffic as one of their focus points [10] and have contributed significantly in Denmark [11]. The Tryg foundation is working with the Council for Safe Traffic in Denmark and taught minors about safety in traffic [12].

Problem Analysis

The RSC states that technology plays an important role in reaching the goals set in the plan of action for 2020. Technology development in the domain of road safety assistance can be divided into three categories [9]:

- Technology that prevents people willing to take risks in traffic intentionally in doing so.
- Technology that aids the road user in acting correctly in traffic.
- Technology that lessens the severity of the accident.

Examples of technology that constraints risk takers in traffic could be the alcohol-breath car lock or equipment forcefully disabling people in speeding. Technology that aids people in traffic could be speed limit alerts, automatic emergency breaks and tiredness detection. Airbags, strong bodywork and cabinet stabilisers are examples of passive technology that lessens the severity of an accident. This project aims to contribute to road safety in the category of aiding the road user to act correctly in traffic. In the 2020 plan of action by the RSC, there are ten prioritized focus areas whereas speeding is a top priority. Speeding is the main factor of fatal incidents in 20% of all traffic accidents and partly responsible for another 21% of accidents in conjunction with both or one of driving under influence and inattentive driving [9]. Due to the severe impact of speeding, the project group sees an opportunity to leverage the user's smartphones and aid the driver in lowering speeding.

Regarding creating technology that aids drivers in traffic to avoid speeding, it is critical to have a road map containing speed limits. Unfortunately, such thing has not been created by the Danish government; only a few municipalities have published the data for their regions, e.g. København Kommune [13]. Commercial maps supplied by e.g. Google, have a significant barrier in their pricing, which excludes them from consideration. Furthermore, Google does not allow their Roads API to be used for insurance risk assessment as stated in *10.4 Restrictions on Unfair Exploitation of the Service and Content* [14] unless an applicable expensive enterprise license is purchased. Open Street Map [15] (OSM) on the other hand, provides a complete road map of Denmark, but with incomplete speed limit data. However, OSM is still considered the most viable solution where the absence of said mapped speed limits must be taken into consideration when designing the technology. This could be countered by crowdsourcing e.g. letting users report speed limits. This further increases the complexity of the system, as validation would be needed to prevent abuse.

Problem Statement

This project seeks to investigate and document how to create a real-time *Mobile Crowd Sensing* (MCS) platform with driver related feedback loops aiming to reduce speeding. The platform should provide the following abilities:

- *Speeding notification* - Notify users in a meaningful way when speeding.
- *Automatic Tracking* - Support automatic tracking, i.e. start and stop trips without user involvement.
- *Speed limit reporting* - Enable users to report road speed limits.
- *App value* - Provide value for the user in the form of driving-related information and a competition aspect.
- *Non-distractive* – The smartphone application must not pose as a distraction while driving.
- *Privacy* - The user's location data is privacy sensitive; therefore, security must be incorporated.
- *Availability* - Available to all drivers with a modern smartphone and internet connection

Furthermore, the platform should support the incorporation of CSR initiatives. The data used to create CSR value must be published in an anonymised, aggregated, and informative format.

Related Work 2

Mobile Crowd Sensing

MCS is a sensing paradigm [16] based on two pillars. The first pillar is the *online community*, which revolves around collecting valuable knowledge from crowds, forming trustworthy answers. The second pillar is the *physical community* which consist of mobile sensors that produce valuable information for the system to collect and process. A formal definition of MCS is provided by Bin Guo et al. [16].

> *"A new sensing paradigm that empowers ordinary citizens to contribute data sensed or generated from their mobile devices, aggregates and fuses the data in the cloud for crowd intelligence extraction and people-centric service delivery."*
> - Bin Guo et al. [16]

The degree of involvement required by the participant can either be participatory, opportunistic or anything in between. Participatory involvement requires interaction to participate in contrast to opportunistic, which revolves around automating the sensing task, easing the participation [17].

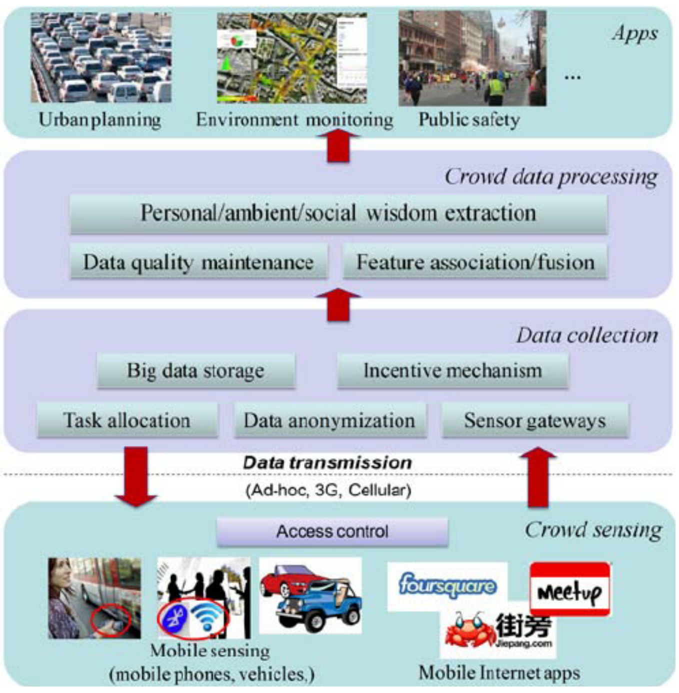


Figure 2.1: A Reference Framework for MCS [16, p.596]

Bin Guo et al. [16] presents Figure 2.1, a proposed MCS architecture consisting of five layers. The layers are *crowd sensing, data transmission, data collection, crowd data processing* and *applications*. The arrows in the figure visualises data flow, which origins from the crowd sensing layer from IoT devices, mobile phones or vehicles. The generated data flows through a data transmission layer to the data collection layer, enabled by network technologies. Several tasks can be undertaken in this layer, such as incentive mechanisms and task allocations which result in data sent back to the crowd sensing layer. Other tasks like storage and data anonymization prepare the data for the crowd data processing layer. This layer applies machine intelligence or logic-based inference strategies to extract high-level intelligence usable by the application layer.

DriveLaB will benefit from the MCS paradigm, by facilitating a speech to text based reporting system for speed limits alongside automatic positioning sensing. Such system will be in the opportunistic range regarding user involvement, which is believed essential for the DriveLaB client application.

Serval systems utilise the MCS paradigm for solving different tasks in different domains. In the domain of monitoring road networks, CarTel [18] and Nericell [19], are two popular proposals.

CarTel makes use of a custom mobile embedded computer running Linux equipped with various sensors alongside a WiFi module. Such device is named a CarTel node and is used for transmitting the measured sensor data to a central server where it can be queried for traffic related usage. Instead of utilising a self-manufactured device as the CarTel node, DriveLaB makes use of the various branded smartphones already in the pocket of an average driver. Even though the two projects differ on multiple aspects, they both have the domain of tracking road users in common.

Nericell aims to monitor traffic conditions of developing regions by utilising sensors already integrated into smartphones. Besides collecting data regarding speed and congestion levels, Nericell can detect honking, potholes, bumps and breaking. Finally, the Nericell system is used in an experiment in Bangalore, India, to reveal the overall performance. Utilising the built-in smartphone sensors is common to both Nericell and DriveLaB. Where Nericell uses multiple sensors such as the microphone, accelerometer, and GPS receiver, DriveLaB centres on utilising only the GPS receiver. Additionally, Nericell scopes their experiment to one city in a developing country, where the system presented in this report will conduct an experiment on a national scale, in a developed country.

Reputation System

Trusting blindly that high quality data will always be supplied by the online communities is not viable [16, p.595]. Therefore, each participant should not be trusted equally, but rather build up trust throughout using the system. To solve this problem, a reputation system will be implemented to judge the validity of the contributed speed limit reports.

Adler and Alfaro [20] presents a reputation system with the purpose of evaluating the trustworthiness of the authors editing Wikipedia, which relies on user-generated content. The reputation system is *content-driven*, meaning the evaluation is solely based on how the authors' contributions fare over time, whereas a *user-driven* reputation system takes input from other users to evaluate how trustworthy a specific user is.

The entire editing history on Wikipedia is available, which is useful to examine how well a contribution has been preserved throughout time. One of the parameters measured is *text life*. This indicates how much of the text contributed by author A is still present after author B has contributed to the same page. The other parameter measured is *edit life*, which indicate how much of the text reorganisation made by author A is unchanged after author B's edit.

Mashhadi and Capra [21] focuses on a real-time ubiquitous crowd-sourced system where data is not exclusively generated from sensors in a passive manner but also generated by users actively providing opinions and perspectives. The context is public transportation where users can contribute with information regarding their trip in real-time, such as traffic accidents, congestions or other dynamic variables that can influence the trip. The usefulness of such application relies on the amount and quality of data generated by the users. While the open nature is a necessity, it is also a threat that can impact the correctness if malicious users pollute the data. The problem is tackled by looking at the users' mobility patterns and the correctness of their past contributions. The mobility pattern for a user is calculated based on where the user normally travels in public places, thereby a report on a location from a user who regularly travels there, has a higher credibility. Additionally, the usefulness of the report's information is evaluated by other users in real-time with gamification techniques. These two factors together form an overall credibility score.

Kantarci et al. [22] focus on the evaluation of users' credibility in the context of MCS systems. The paper discusses existing approaches to quantify crowd-sensed data trustworthiness based on statistical and vote-based user reputation scores. A new metric called *collaborative reputation scores* is proposed, which draws from the strength in the statistical and vote-based user reputation scores. The statistical reputation based MCS calculates the users' credibility in a *centralized* manner, as it is the platform itself which evaluates each user. This is done by statistically looking at the percentage for which a user has provided a correct reading. The vote-based approach calculates user reputation in a *decentralized* way. The users who are assigned to common sensing tasks form a network, where they can vote for each other, to build up their reputation.

The reputation system implemented in this project apply elements from the papers described in this section. However, as the context is speed limits on road networks, the system also considers road properties, which can influence the credibility of a speed limit report. Additionally, information regarding the geographical location of where users typically drive is taken into account.

Map Matching

As this project uses speed limits to calculate the score of a trip, it is necessary to know which stretch of road the driver is currently on. Therefore, it is vital to map the raw GPS input to a virtual representation of the road segments. Furthermore, the system presented in this report will focus on real-time map matching, since feedback based on their current road segment location is provided. Therefore, literature concerning post processing map-matching (MM) algorithms are not considered.

Over time, numerous MM algorithms have been developed to improve accuracy and efficiency leaving behind an extensive collection of MM algorithms. Therefore, articles such as [23] have been written to supply a quick overview of this field. It is worth mentioning that [23], solely focuses on real-time MM algorithms, however, this fits the project's use case. The article groups the MM algorithms into three categories, namely, simple, weight based and advanced. It is argued that simple MM algorithms do not provide the necessary accuracy where advanced MM algorithms add too many calculations, increasing the processing power requirements as well as decreasing the readability of the algorithm. However, weight based MM algorithms are found to perform well regarding accuracy and efficiency.

Bristow et al. [24] have developed one of the weight based algorithms presented in [23] showing promising results. The algorithm consists of the three steps, initial MM, MM on a link, and MM at a junction.

- *Initial MM* – Calculates a total weight for each segment intersecting an area represented by an ellipse which radius is equal to the accuracy of the GPS point in metres. In this step, two different weights are used to assess the probability of a road segment. Firstly, the closer the heading of a GPS point is to a road segment's bearing, the more likely it is to be the right road segment. Secondly, the closer a road segment is to the GPS point, the more likely it is to be the right road segment.
- *MM on a link* – Since a road change can only happen at a junction, it is not necessary to consider other road segments than the previous mapped, as long as a certain distance to the downstream junction is present. Therefore, this step only keeps track of the distance to the downstream junction, as well as if the car has turned.
- *MM at a junction* – If the car is near a junction or has turned, a procedure similar to the initial MM, will be executed, but with two additional weights. These are both topological weights, where the first considers if a road segment is directly connected to the previous mapped road segment. If it is directly connected, it is more likely to be the right road segment. The second weight considers if a transition from the previous road segment to each of the candidates is legal to make. If not, it is less likely to be the right road segment.

The value of both heading and distance weights can be any real $\mathbb{R}$ in the interval $[0, 1]$ with 0 and 1 inclusive where the two topological weights must evaluate to either 0 or 1. The degree of how well a point maps to a road segment regarding heading, distance, linkage and turn restrictions is determined by the magnitude of the weight. Furthermore, each weight can be multiplied by a weight coefficient, regulating the importance of the weight given certain environmental properties. [24] adjusts these weight coefficients based on the operational area in respect to the three categories, urban, suburban and rural.

Performance Testing

As an MCS system is developed in this project, the value of the system is correlated with the amount and quality of data gathered. To gather a high quantity of data, a sizeable user base is required. This place demands on the system's performance and its ability to scale when the user base grows. Therefore, ideas for testing system performance and scalability is investigated through related work.

Weyuker and Vokolos [25] discuss an approach to performance testing. They mention the importance of having testing goals in mind when designing the tests. Examples of testing goals can be, the number of users the system can handle or identifications of hardware or software bottlenecks. They also outline the typical steps of a performance test, which are described in the following [25, p.1155]:

1. Identify the software processes that directly influence the overall performance of the system.
2. Select essential input parameters that influence the performance of the system.
3. Provide realistic values for these parameters by collecting and analysing existing usage data.
4. If there are not historical data available for some parameters, then make reasonable estimations of values based on requirements for developing the system.
5. If a range of values for a given parameter exist, then choose values from the range that can expose useful information about the performance of the system.

Predic and Stojanovic [26] evaluates the performance of their system using crowdsourcing by simulating numerous car trips concurrently and measuring the amount of data transferred between the smartphone and server. Parts of the heaviest computational tasks are offloaded to the smartphone, which might require a lot of CPU power from the smartphone, therefore battery usage tests are conducted as well.

This report does not contribute to the topic of performance testing. However, the work described in this section serves as inspiration for how performance tests are conducted in this project, which is presented in Section 5.1.2 - *System Performance Testing.*

Usage Based Insurance

Usage Based Insurance (UBI) is considered related work as it is a significant promoter of systems like the one in this project. The landscape, parties involved and benefits of UBI is described in [3, p.2-5]. An interesting new initiative has emerged since, as Tryg Insurance is launching UBI in Denmark in the summer of 2017 [27]. The initiative, however, received negative publicity by the public and the press which has named the initiative "surveillance".

> *"I think it is really positive, for example younger people has the opportunity to get much cheaper car insurance while getting aware of how to drive and the risks involved with driving" - Morten Hübbe, Tryg Group Executive.*

Baecke and Bocca recently published an article [28], investigating the value of vehicle telematics that leads to UBI. The article is based on data from 6984 customers at a European car insurance company in the years of 2011 until 2015. These customers all accepted having an In-Vehicle Data Recorder (IVDR) installed in their car, which tracked their everyday driving. The IVDRs sensed more than 230 million kilometres in total. A literature overview is included with studies that investigate different methods to analyse driving in terms of risk. They compare by research scope, sample data, observation period, predictors and analytical techniques, revealing their article as most ambitious regarding predictors included, thus being complex.

The interesting part of the article is that besides well-known insurance premium related standard parameters in Denmark like driver's age, driving experience, the make, model and age of the car, and data of past claims, they also consider *telematics data*. This data includes kilometres driven which are subdivided based on time and location and is cross-examined with *expert-based data*, e.g. experts know that night trips in weekends have a significant impact on accident risk. The article processes the collected data using different machine intelligence methods to find the optimal model to perform a risk assessment.

The study finds that combining traditional insurance risk calculation with the telematics data is the most optimal as they capture different risk elements, however when initiating UBI in an insurance company, it is advisable to first focus on *Pay-As-You-Drive* plans where the premium is based on kilometres driven. This is due to the risk of an accident being highly correlated with amount of time spend on the road. As data start accumulating, the company can improve their risk assessment by including time and location as the next step. Lastly, the study found that only 3 months of customer data is enough to assess the accident risk factor [28, p.19].

Analysis & Design 3

The analysis and design process of the MCS system DriveLaB are based on the related work of MCS presented in Chapter 2 - *Mobile Crowd Sensing*. Namely, the five layer architecture shown in Figure 2.1 is adopted with minor differences.

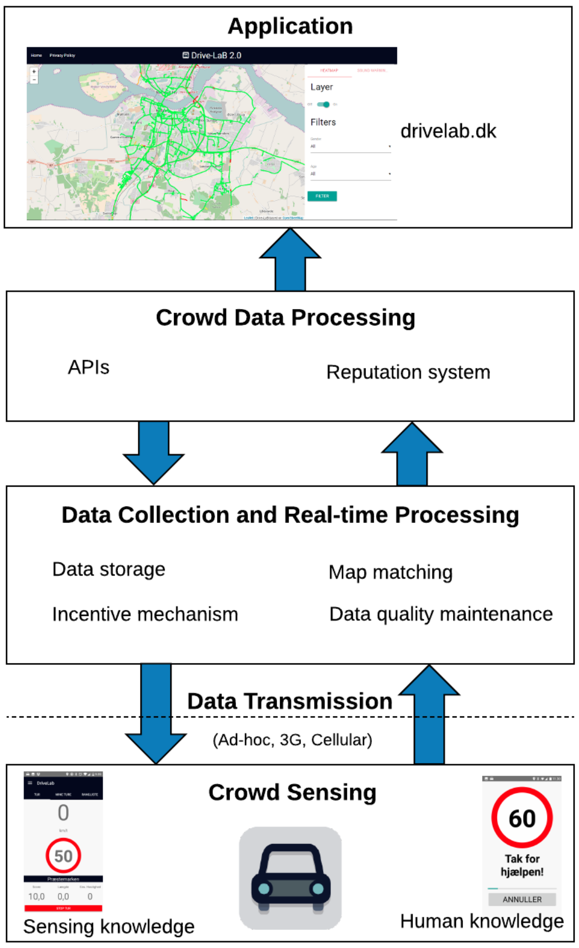


Figure 3.1: DriveLaB MCS Architecture

The most significant difference is the extension of the *Data collection* layer also to include real-time processing. This processing is needed to supply real-time driving information feedback to the client application crowd sensing layer. Furthermore, the *Crowd Data Processing* layer sends data back to the data collection and real-time processing layer, indicated by the arrow. This is to update the data storage with updated speed limits reported by users and processes by the reputation system.

The design decisions are grounded in the project group's experience in designing and experimenting on the system presented in [3], along with feedback from real world users. Section 3.5 - *Applications* is an added component to visualise the data collection usage, demonstrate the CSR value and it is an integrated MCS component. Table 3.1 emphasises the evolution from the work conducted in [3] to the new and enhanced design presented in this report.

Design element	**Previous system**	**New system**
Client technology	Ionic 2 using JavaScript, Typescript and Java	Utilise Microsoft Xamarin and rewrite code to C#
Client platform	Android	Android and iOS
Handling missing speed limits	Show speed limit as missing and omit the segment in score calculation	Use a heuristic algorithm and allow user-driven speed limit correction by utilising speech recognition
Speed limit reporting	No option to report wrong or missing speed limits	Users can report speed limit corrections
Data transfer encryption	No encryption	Encryption
Map matching	Naive simple "Nearest-Road" approach	Weight and topological based algorithm from literature [24]
Automatic tracking	Garmin GLO and Kontak.io Beacon device	Generic utilisation of Bluetooth units pre-owned by drivers
Scoring system	Based on speed exceedance	Similar approach but more efficient calculation
Short feedback loop	Speed, speed limit and speed exceedance sound notification.	Speed, speed limit, speed exceedance sound notification, road name, score, trip length and average speed
Medium feedback loop	No feedback.	Auditory acknowledgement of good driving.
Sound notification data	Notification type and sound settings	Notification type, sound settings and speaker volume
Leaderboard	Based on all data	Filter by active users
APIs	Private API	Private and Public API
API versioning	No versioning	Header versioning

Table 3.1: DriveLaB System Version Comparison

Comparing Table 3.1 to the future work section of [3, p.57], reveals that every point of future work is being considered directly or indirectly in the new system design. Besides the quick overview of improvements, each subject is further elaborated in the MCS component to which they belong. The improvements contribute to stability, performance, and user experience.

A significant change in the functionality that spans across several components is to handle the lack of speed limit data in the road network differently. In [3], a score calculation was based strictly on road segments driven where the speed limit was known, thus disregarding speed exceedance where the speed limit is unknown. Users expressed frustration when facing lacking speed limits and it had a significant negative impact on the feedback loop. Especially as only 49% of the data collected was on roads with known speed limit [3, p.57]. Instead, the intention is to use a heuristic algorithm where speed limits are not available and allow users to report adjustments, hereby improving the dataset of speed limits. This design change impacts all layers of the architecture and is described in the respective sections.

Three aspects in [3] are retained in the new system with no changes applied as no feedback or experience revealed dissatisfaction. These are *auditory speeding notifications, speed exceedance scoring system* and *driving rating smileys*.

3.1 Mobile Crowd Sensing (Client Application)

The entire *client technology* used to create the client smartphone application is switched from Ionic 2 [29] to Microsoft's Xamarin platform [30]. This is a consequence of the experience gained with Ionic 2 as platform specific features like services on Android [3, p.56] are exceptionally hard to debug in the Ionic 2 platform. The asynchronous nature of non-OOP JavaScript in combination with synchronous OOP Java further complicated the intercommunication between the languages. Also, the lack of a native design look and feel due to it essentially being a browser rendered web page raised concerns about the viability. It was moreover also to accommodate the feedback received from LB Insurance stating that an iPhone version of the application is a necessity for future tests, which would require the use of the Objective-C language. Switching from Ionic to Xamarin means switching programming languages from Javascript, Typescript, and Java to C# and switching mark-up languages from HTML to XAML. However, it eliminates the necessity to learn the Objective-C language to develop the iPhone version of the application and enables one code-base for multiple mobile platforms.

Abstracting away the fact that the road network dataset is incomplete in terms of speed limits and use a heuristic algorithm as conducted in Section 3.3 - *Data Collection and Real-Time Processing*, entails the functionality for the users to report wrong speed limits when encountered. This poses a challenge as any physical phone interaction is illegal, and therefore, speech recognition is considered the only viable option. Unfortunately, [3, p.57] do not state if auditory feedback does have a reductive effect on speeding as the smartphone volume is not tracked. This is a necessity to include in the current application.

Automatic tracking was named the biggest technological challenge in the feedback from users, and the usage of multiple devices proved a substantial task to support programmatically. Therefore, the idea described in [3, p.56] is to utilise common Bluetooth devices already present in users vehicles, which represents a major but necessary change to accommodate automatic tracking. In [3], an underlying process is always running in the background while scanning for the external devices. The new logic is depicted on Figure 3.2.

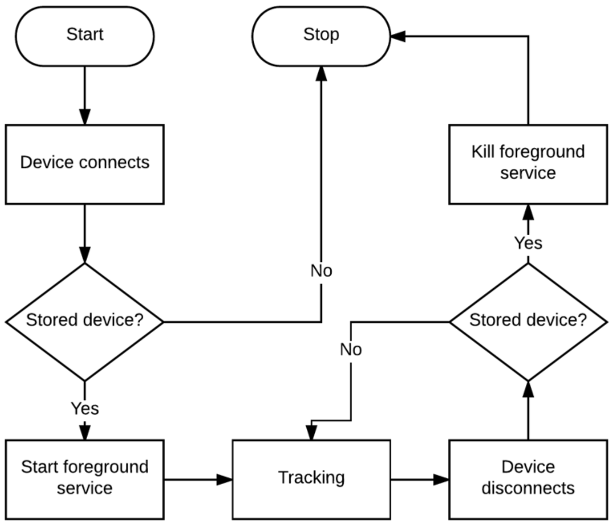


Figure 3.2: New Automatic Tracking Flowchart

Utilising *Broadcast Receivers* [31] to receive a signal when a *device connects* via the Bluetooth protocol [32] enables the application to wake and take action, and thereby eliminating the need to be permanently running as a service. The device connected can then be compared to what the user has chosen as their car-installed device. The *Stored device?* decision point checks if the connecting device and chosen automatic tracking matches. If not, the execution should stop, and on the contrary, if it matches, the foreground service [33] should be started and commence tracking. Similarly, if the system is in a tracking state, devices disconnecting from the phone via the Bluetooth protocol is compared to the stored device. If it is the stored device that disconnects, the tracking is stopped, and the foreground service is terminated. The comparison of the connected or disconnected device to the stored device is the same functionality. However, the reaction differs depending on if the service is already tracking or not. Hence, the two decision points named *stored device* in Figure 3.2. There are four conditions to this approach of automatic tracking:

1. This solution only applies to Android smartphones as this is highly platform specific and a similar solution has not been discovered for Apple smartphones.
2. The stored Bluetooth device must be mounted in the car and power on and off with the car.
3. The stored Bluetooth device must automatically connect to the smartphone when powered on.
4. Bluetooth and GPS on the smartphone must always be enabled.

Conditions two and three often applies for Bluetooth devices meant to be installed in a car. Condition four is considered a substantial threat against the logic, as smartphone users might turn off these functionalities to preserve battery.

3.2 Data Transmission

Similar to [3], the new system needs to facilitate real-time updates of information regarding the user's driving, which means that a bi-directional real-time data transmission technology is needed. To improve code transparency, control over events and reduce communication, the framework SignalR [34] utilised in [3] is not implemented. Also, SignalR for ASP.NET Core is yet in a development state and to date has no stable releases. However, the experience of SignalR utilising the Websocket protocol for data transmission proved effective.

Websockets are a TCP-based [35] protocol, which is preferable over UDP [36] that do not provide any guarantee of delivery nor ordering. These guarantees are needed to ensure that locations are received and arriving chronological.

Therefore, the bare WebSocket [37] technology is used, which leaves coding tasks like handling connections, disconnections, message parsing up to the coders. Furthermore, the sensitive data is to be encrypted in the transport as specified in Table 3.1. Figure 3.3 illustrates the data transportation design.

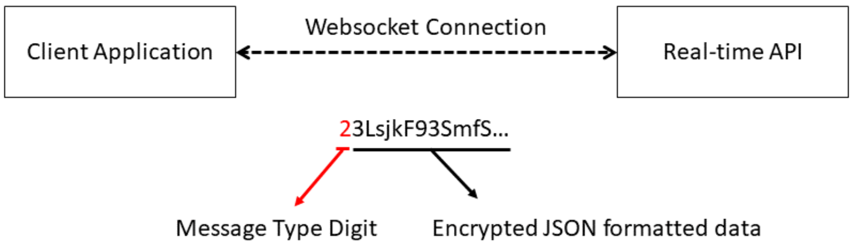


Figure 3.3: Data Transmission in DriveLaB

The *Client Application* and *Real-time API* communicates through a Websocket communication protocol using *data transfer objects* (DTOs) [38]. The communication is bidirectional, as depicted by the dotted line. The transmitted data is in the format of a one-digit prefix message type encoding followed by the actual data as a JSON formatted object. This structure enables the receiving platform to identify what type of data is hiding behind the encrypted string and to decode properly. Naturally, the one-digit prefix indicating the message type entails a restriction in the number of message types available. This is sufficient as the system is not designed to exceed the ten available message types. Furthermore, the addition of message types carry a significant amount of code both on the client and server side besides just defining the new message type, and it would be trivial to alter the logic to look at several digits or something more complex when exceeding the message type limit.

3.3 Data Collection and Real-Time Processing

This layer receives the mobile sensor data, which includes GPS locations and audio configurations of the smartphone as well as speed limit reports contributed by users as described in Section 3.1 - *Mobile Crowd Sensing (Client Application)*. As seen in Figure 3.1, four components reside in this layer:

- *Incentive Mechanisms* describes how users are encouraged to participate by processing the data in real-time, thereby providing driving related feedback during the trip.
- *Map Matching* deals with detection of where users are located to further enhance the data.
- *Data Quality Maintenance* helps filter away polluted data.
- *Data Storage* saves the data once it has been processed where further analysis can be conducted.

These components will be further elaborated in their respective sections.

3.3.1 Incentive Mechanisms

Users can be motivated by different incentives, such as monetary, entertainment, social, ethical or interest [16, p.595] where this project features the last three. The ethical motivation aspect is expressed in the concept of the application, which essentially attempts to help users drive safely, thereby avoid putting themselves and others in danger.

To address users who are interested in driving-related information, an implementation of real-time feedback throughout the trip is provided. As shown in Table 3.1, the design element *Short feedback loop* only consisted of speed and speed limit information. In this project, this feedback loop is extended by road name, score, trip length and average speed. Furthermore, [3] only involved the short feedback loop, which notifies the users when they are exceeding the speed at certain percentages, and the long feedback loop, which provides detailed information about the trip afterwards. As design element *Medium feedback loop* shows, this project provides the addition of a medium feedback loop. This is intended to provide positive feedback to the users, whenever they have not exceeded the speed limit for a specified distance.

The social aspect manifests in the design element *Leaderboard*, where users can compete to become the best driver, based on a total trip rating assigned to each user. Ideally, it should be possible to form private leaderboards where users can compete against friends and family, however, this is not the focus of this report. A leaderboard is implemented in [3]. However, it does not account for inactive users, which is a problem. This needs to be solved since it does not motivate users to keep using the system and improving once a satisfactory performance score has been reached. An approach to this issue is to have seasons where the leaderboard gets reset. Depending on how often the leaderboard resets, it is mostly active users who are present. Another approach to address this problem is by implementing a sliding window, which only considers active users. Whether a user is active, is based on the distance driven within a certain number of days. It is only the score of trips within a recent time interval which are considered. This avoids punishment and unnecessary discouragement of people who have performed badly in the past. It is decided to implement the sliding window, instead of leaderboard seasons, as it is believed to be superior in this context since it captures the active users at any given point compared to seasonal leaderboards where inactivity might occur before leaderboard resets.

3.3.2 Map Matching

As seen in Table 3.1, the design element *Map matching* handles the detection of which road the user is driving on. A simple approach assuming that the user is always driving on the road nearest to the GPS point is utilised by [3]. The MM algorithm presented in Chapter 2 - *Map Matching* is used in this project, which improves the detection and thereby enhancing the correctness of feedback sent to the user. Specifically, it partially solves a problem from the last project where a user was map matched to the wrong lane going in the opposite direction. It handles this by using the GPS heading which indicates in which direction the user is travelling. Additionally, the GPS locations are projected onto the map matched road, which allows much easier debugging of the map matching algorithm in addition to a more precise real-time calculation of distance driven for the users.

Even with a more accurate MM algorithm, it is not always possible to tell the speed limit of a road, due to the partial lack of speed limits posed. As seen in design element *Handling of missing speed limits*, this issue is dealt with in the last project by not displaying a speed limit to the users and omitting using that road segment in the calculation of performance score. In this project, it is approached differently by implementing a heuristic algorithm that calculates the speed limit of a road where it lacks, based on the road type. There are three main types of roads in Denmark, which is a motorway, rural road, and city. They have the speed limits 130, 80 and 50 km/h respectively[39]. The advantages of using this approach is the possibility of providing a performance score to the user, which is based on the entire trip, including roads where the speed limit is not contained in the dataset. Additionally, the application appears more functional to the user, as it will for the most part display a speed limit, even when it is unknown. The disadvantage of this approach, is the possibility of irritating users by punishing their performance score wrongly or playing a sound notification at an undeserved time.

An important design choice is the decision of where the digital map containing speed limits should be located, as it heavily influences both functionality and performance of the system.

Map server-side	**Map client-side**
Easy propagation of speed limit changes	Faster feedback response time
Less battery consumption	Better scalability
No additional data storage usage	
Simpler to implement	

Table 3.2: Advantages of Storing the Map Server-side or Client-side

Table 3.2 shows the advantages of storing the digital map containing the speed limits on the server-side and client-side respectively. The advantages of storing the map, or chunks of it, client-side is the faster response time, since MM can be done on the phone. This removes network delay and the need to accumulate GPS locations before sending to the server for processing of GPS locations. Additionally, this ensures that most features related to tracking still works without internet connection. The system becomes more scalable as the heaviest processing is off-loaded to the smartphone.

Storing the map server-side makes it overall simpler to implement, as changes made to the map will take effect immediately. This is more difficult when it is stored client-side, as changes made to the map needs to be synchronised across all users. This also requires the application on the smartphone to use a lot more data storage. Having the map server-side will also entail performing the heaviest processing on the server. While this makes the system less scalable, it potentially makes the smartphone application less battery consuming. It was decided to store the map server-side, because it provides simplicity and creating a system able to support a huge user base has not been the focus of this project.

3.3.3 Data Quality Maintenance

In an MCS system, the sensed data received should be filtered, as some of it may be redundant or sensed under inappropriate conditions [16, p.595]. This challenge is approached by removing all data related to a trip which is deemed invalid. A trip is considered invalid if it is under 500 metres, this helps ensure that trips started by e.g. accident are not stored in the database. To filter out GPS locations which are of too low quality, the map matching algorithm ignores GPS locations if it is map matched to a road segment, where the distance between the GPS location and road segment is over 160 metres. The 160 metres origins from empirical investigation implying that the algorithm must have identified a wrong link at this distance [24, p.676]. Other methods to filter away low-quality GPS locations should be implemented as environmental factors such as sky blockage and atmospheric effects can affect the quality of GPS locations [40, p.817], however, this has not been a focus in this project.

3.3.4 Data Storage

To accommodate the new system goals outlined in Table 3.1, the *Data Warehouse* model presented in this report has only few similarities to the one presented in [3]. A complete exposition of the new and improved database schema is provided in Appendix A - *Data*

Warehouse Schema. This section aims to highlight and discuss the most significant design alterations.

Dimension Tables

The application developed in [3] autogenerated a user when the application opened for the first time. Unfortunately, this signified, acquiring a new phone would entail the data accumulated with that profile is lost. To solve this problem, it is decided to enable the user to create a profile with credentials such as username, email, and password. Logging in on an arbitrary phone with the right credentials will then display the associated data. Date of birth (dob), gender, time of creation are metrics included to permit a more detailed view of the target group as well as when users tend to sign up (after campaigns, weekends, after broadcasted traffic accident, etc.).

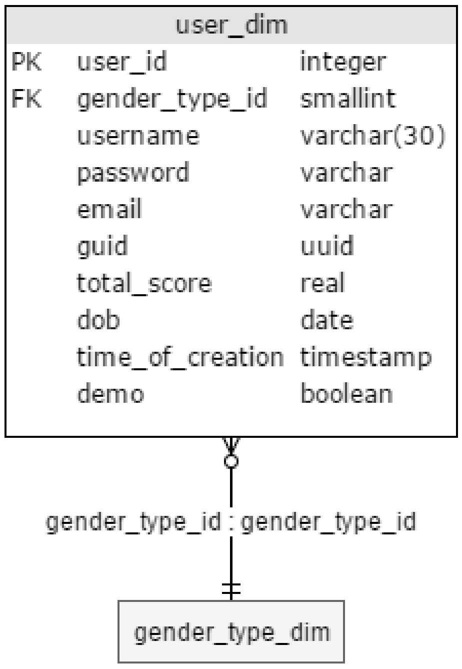


Figure 3.4: User Dimension Table

In accordance to [40] environmental variables such as weather, is pinpointed as UBI fundamentals. Furthermore, as mentioned in Section 3.3.3 - *Data Quality Maintenance*, the weather may have a negatively impact on the GPS quality. Also, meteorology data allows investigating if weather has an impact on road users' behaviour. Based on these three arguments, it is decided to include a weather table, depicted in Figure 3.5.

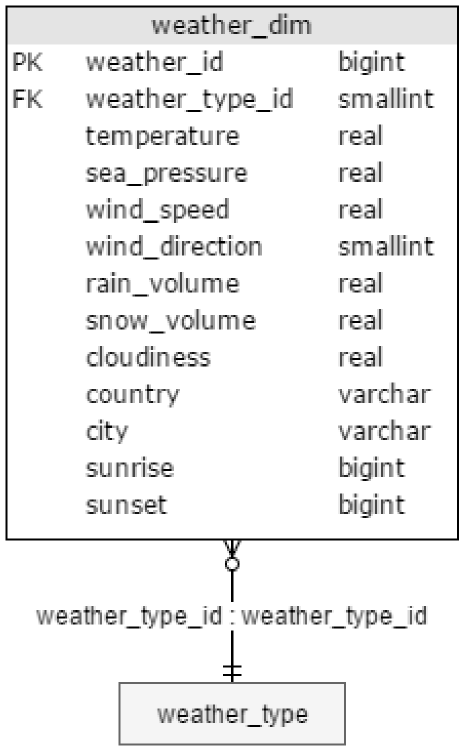


Figure 3.5: Weather Dimension Table

Only two device types (Garmin GLO and Kontak.io Beacon) are supported by the system developed in [3]. As mentioned in Section 3.1 - *Mobile Crowd Sensing (Client Application)*, it is decided to support multiple devices commonly found in vehicles. Therefore, a new table is introduced, replacing the car table presented in [3]. This table aims to reveal the diversity of devices that are used for the DriveLaB application. The different aspects are *device_ type_ id* (Bluetooth Low Energy, Bluetooth classic or both combined), *mac_ address*, *class_ of_ device* (hands-free, wearable headset, headphones etc.), and *device_ uuid* (often include brand and model details).

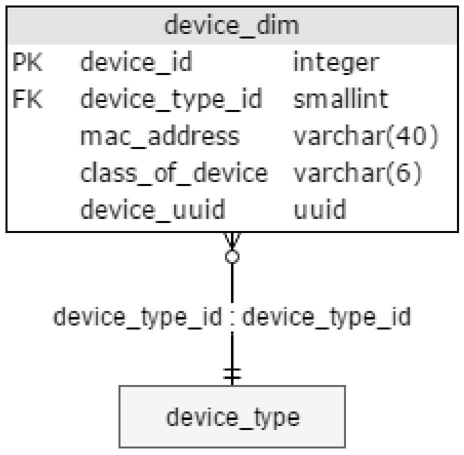
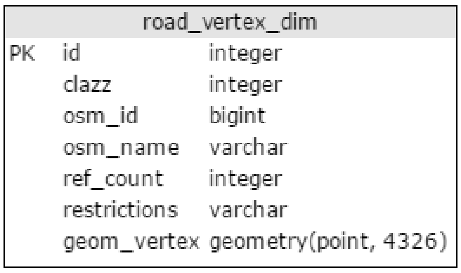

device_dim
PK device_id integer
FK device_type_id smallint
mac_address varchar(40)
class_of_device varchar(6)
device_uuid uuid
device_type_id : device_type_id
device_type

Figure 3.6: Device Dimension Table

The MM algorithm presented in Chapter 2 - *Map Matching* and discussed in Section 3.3.2

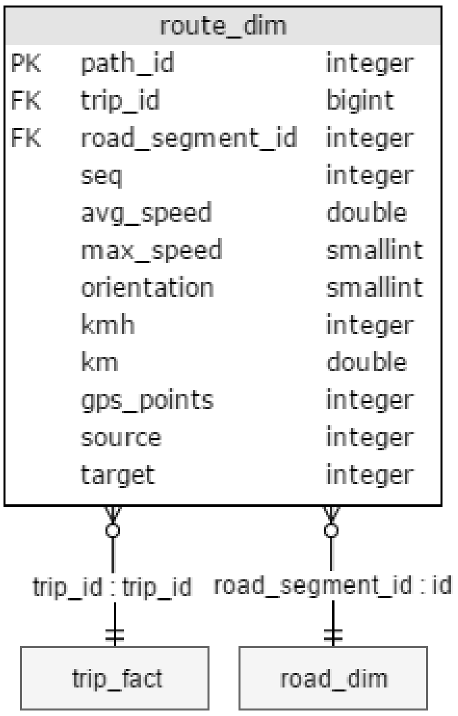
- *Map Matching* is, besides distance and heading, based on two additionally topological properties. These properties are *connectivity* and *turn restrictions*. Figure 3.7 depicts the table responsible for storing the necessary topological information.



road_vertex_dim		
PK	id	integer
	clazz	integer
	osm_id	bigint
	osm_name	varchar
	ref_count	integer
	restrictions	varchar
	geom_vertex	geometry(point, 4326)

Figure 3.7: Road Vertex Dimension Table

It would be inefficient to consider all GPS points when drawing paths to a map. Therefore, a route dimension table is added, considering only road segments. The *seq* column will make sure the correct road segment sequence is never lost, where *kmh* and *km* will store the speed limit and the distance of the road segment respectively. This way, if the speed limit changes over time or the road segment gets altered, it will not influence old trips utilising said road segments. *Avg_ speed*, *max_ speed*, *gps_ points* are all aggregations potentially based on multiple GPS points.



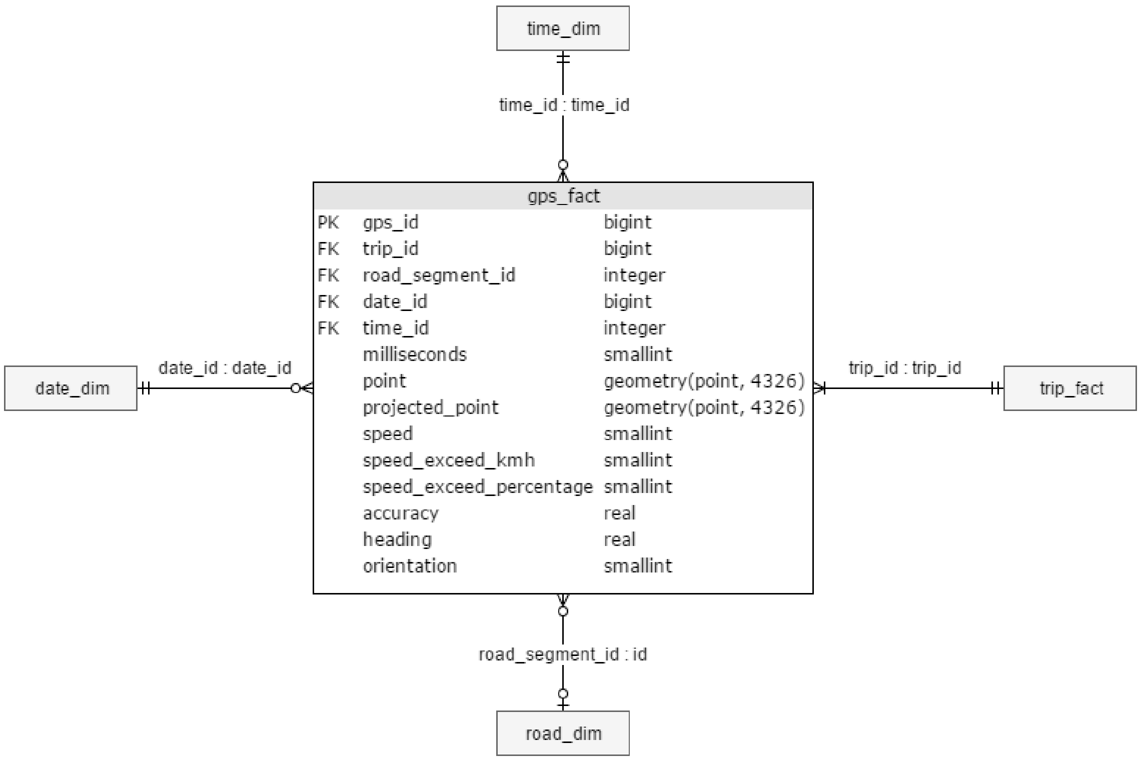
route_dim		
PK	path_id	integer
FK	trip_id	bigint
FK	road_segment_id	integer
	seq	integer
	avg_speed	double
	max_speed	smallint
	orientation	smallint
	kmh	integer
	km	double
	gps_points	integer
	source	integer
	target	integer

Figure 3.8: Route Dimension Table

Fact Tables

Telling the sequence of two GPS points measured in the same second if only the time dimension table is used, is not possible. This problem is not handled by the schema presented in [3]; however, the new schema solves this by adding a milliseconds column.

Additionally, the projected points mentioned in Section 3.3.2 - *Map Matching* is stored in the GPS fact table. Accuracy, heading and orientation is also data not previous stored, which allows the MM algorithm to iterate through trips multiple times which is beneficial for debugging purposes.



	gps_fact	
PK	gps_id	bigint
FK	trip_id	bigint
FK	road_segment_id	integer
FK	date_id	bigint
FK	time_id	integer
	milliseconds	smallint
	point	geometry(point, 4326)
	projected_point	geometry(point, 4326)
	speed	smallint
	speed_exceed_kmh	smallint
	speed_exceed_percentage	smallint
	accuracy	real
	heading	real
	orientation	smallint

Figure 3.9: GPS Fact Table

It is in [3] inconclusive whether the auditory speeding notifications got heard or not. This is due to not tracking the smartphone volume when playing the sounds. The sound level will, among other parameters, be stored in the sound fact table for later analysis.

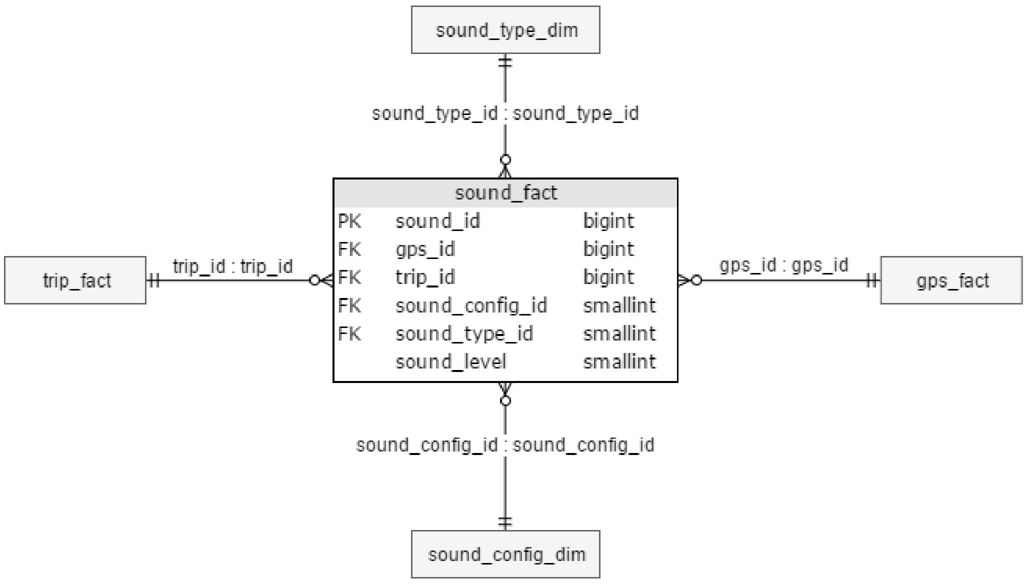


Figure 3.10: Sound Fact Table

Figure 3.11 shows the table containing all the speed limit reports contributed by the users. The column *correct_ report* is used to indicate whether the speed limit report is determined to be correct or incorrect. This table also tracks which reports have influenced the decision to change the speed limit for a road, which the column *road_ history_ id* manages.

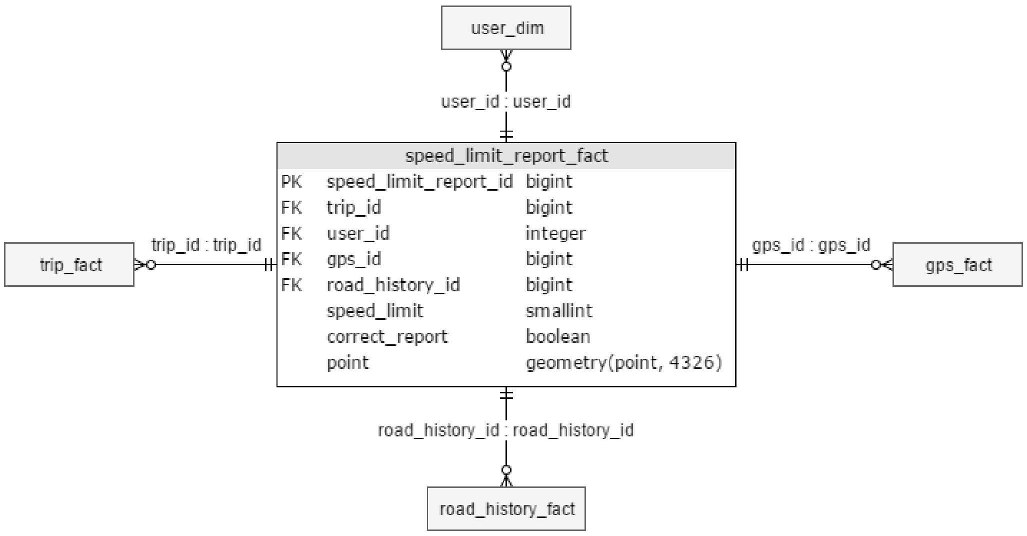


Figure 3.11: Speed Limit Report Fact Table

The table depicted in Figure 3.12 stores all speed limit changes made to the roads. This includes the dates when the changes happened and what specifically the speeds were changed from and to. This allows for tracking of how the speed limit has evolved for a given road.

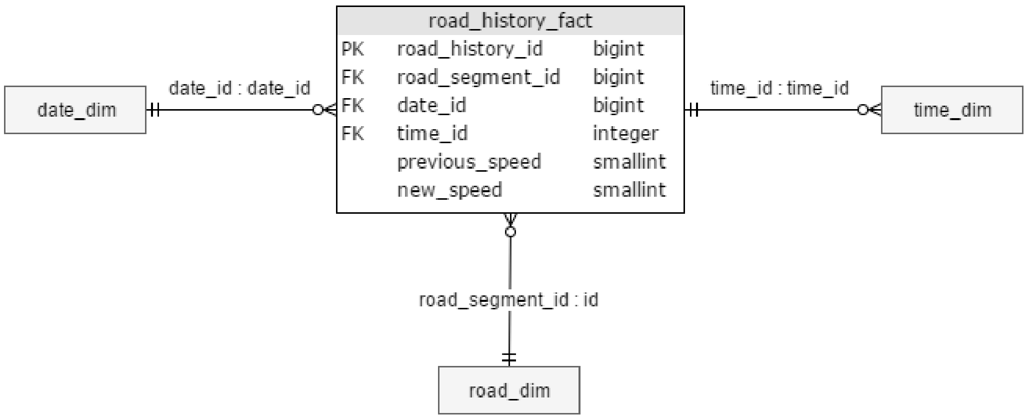


Figure 3.12: Road History Fact

3.4 Crowd Data Processing

This section revolves around utilising the data collected and stored in Section 3.3 - *Data Collection and Real-Time Processing* to deduce new valuable information. The *Reputation System* will evaluate the trustworthiness of speed limit reports contributed by users where *APIs* can be used to support data interchange between the data warehouse represented in Section 3.3.4 - *Data Storage* and the applications discussed in Section 3.5 - *Applications*.

3.4.1 Reputation System

As described in Section 3.1 - *Mobile Crowd Sensing (Client Application)*, the smartphone application provides the functionality for users to report speed limits. This is intended to be utilised if the users encounter incorrect speed limits. However, users may report an incorrect speed limit due to an accident or malicious intent. This can pollute the dataset with incorrect data and thereby affect the performance score for other users driving on that road segment inappropriately.

To approach this challenge, the speed limit reports are processed to determine if the road speed limit should be changed. As described in Chapter 2 - *Reputation System*, elements from those contributions are used in the design of the reputation system implemented. Specifically, the system presented in this report base the reputation of users on how their speed limit reports fare over time. This works similarly to the proposed reputation system by Mashhadi and Capra [21], which bases the reputation of users on how their Wikipedia contributions fare over time.

Users who have contributed with speed limit reports for a road segment which has caused a change in speed limit will receive a better reputation. However, if new speed limit reports for that same road segment is submitted, the speed limit will change again. The time interval between these two changes determines if the first group of people should be punished by having their reputation score changed negatively, if it is within a short time interval or leave them unpunished if the time interval is long.

It is both the latest and past submitted reports which are used in the calculation of users' reputation score, similar to the way Kantarci and Mouftah determine user reputation [41, p.362]. As described in Chapter 2 - *Reputation System*, [21] uses the mobility pattern of a user to influence the reputation score. This element is applied in this system, where the credibility of a speed limit report is influenced by not only the reputation of the user but also the geographical location of where the speed limit report was captured. If the geographical location of the speed limit reports is placed within the area where the user normally drives, the credibility of the report is affected positively.

The road type is also considered as it influences what speed limit ranges are possible e.g. a speed limit report suggesting changing a motorway road from 130 km/h to 50 km/h is unlikely.

Another approach that could have been used instead to detect the correct speed limit for the roads, is to observe the speed at which users drive. Based on the speed of how users drive in general, it might be possible to deduce the correct speed limit for that road. This has the advantage of not requiring users to report a speed limit manually. Instead, the GPS data collected by tracking with the client application can be used to determine this. However, it has the disadvantage of requiring a sizeable amount of data to implement, which has not been a possibility for this project.

Bin Guo et al [16] mentions the combination of human and machine intelligence where the system *DietSense* is referred to, which utilises both image processing techniques and manual image review by humans [16, p.596]. It might be interesting to investigate in the future, how the same approach could be applied in this context where speed limit reports from humans are used together with machine learning techniques that can derive the speed limit of roads based on GPS data.

3.4.2 API

APIs are used as an interface for communication between system components. In this report, three different API exposure levels are presented, namely *public*, *protected*, and *private*. These levels can be used to enforce policies in regards of sharing person identifiable data. Furthermore, different techniques for API versioning is outlined.

Access Levels

Public	**Protected**	**Private**
This access level should be open for the general public and precautions to deny access to person identifiable data should, therefore, be enforced.	Offering an interface for partners has many use cases. Therefore partner credentials should be provided to receive the desired data. If data is exchanged in accordance with the terms of use, anonymity should not be an issue	An API for internal use only can be applicable for transferring data between system components. This denotes that the API should not be available for the public nor should it be for partners, to avoid privacy concerns.

Table 3.3: API Access Levels

In this project, both a public and private REST API is implemented to facilitate a website- and smartphone application respectively. Implementing these two API access levels should prove the viability of the protected API. The public API is elaborated further in Section 3.5 - *Applications* where the private API is designed almost identical to the one presented in [3].

Versioning

Different methods are used for versioning REST APIs, where the most common ones are displayed in Table 3.4.

Name	**Type**	**Example**
URI path	URI	domain.com/2.0/resource
Domain name	URI	apiv2.domain.com/resource
Query parameter	URI	domain.com/resource?apiversion=2.0
Media type	Header	Accept: application/apiv2.0+json
Custom header	Header	X-API-Version: 2.0

Table 3.4: API Versioning Methods

Even though implications regarding versioning methods are well known [42][43][44], the impact of these implications are more biased. Therefore, a thorough discussion of pros and cons is deemed out of scope for this project. However, both [45] and [43] argues that using URIs to implement versioning in REST APIs, violates the "one resource, one URI" concept, and is therefore not used in this project. Additionally, it is assessed that header fields should offer a clear and narrow set of options which is why using the *Accept* header field for versioning is deemed not optimal. This leaves the custom header option, which is also the one chosen for versioning the public and private REST API in this project.

3.5 Applications

In accordance to [16], MCS can leave a variety of applications and services open for development. This section will be investigating CSR related opportunities hiding among these software solutions. To narrow the scope, it is decided to focus on three fields, namely *Map Contribution*, *Improvement of Road Safety* and *Improvement of Road Infrastructure*.

Map Contribution
Speed Limit Map - Google and Microsoft both own commercial online map services [46] [47] that lacks a 100% speed limit coverage. Therefore, producing a map with speed limits which are made freely available to benefit the community of open data could benefit a company by producing positive press coverage.
Improvement of Road Safety
Speeding Visualisation - Offering free access to a map that highlights road segments where speeding is common. This information can be used by the police to better assess where to place speed controls, and thereby improve road safety.
Danger Zone Detection - Locating areas with harsh breaking could also be an example of providing valuable information. The road authorities could place e.g. chicanes or roundabouts in these areas to improve the road safety.
Driver Feedback - As done in [3], the client application itself could contain features, such as speeding notifications and scoring for keeping within the speed limit. This may prevent accidents related to speeding and thereby improve road safety.
Improvement of Road Infrastructure
Congestion Level Visualisation - A heatmap showing heavily trafficked road segments can help road authorities visualising where to expand the road network or placing bypass roads to relieve stress. This will help improve the traffic flow and eventually reduce the travel time for road users.

Table 3.5: CSR Ipportunities in MCS

Speeding Visualisation is one of two elements in Table 3.5 that will be implemented in this report. However, *Speeding Visualisation* is the only new contribution compared to [3], a reimplementation of *Driver Feedback* is also included. Development of *Speed Limit Map*, *Danger Zone Detection* and *Congestion Level Visualisation* is thereby not included in this report, avoiding too broad a scope.

Speeding Visualisation is designed to be an API offering the possibility of querying the database through HTTP GET requests. It follows the public API policy presented in Table 3.3 and consists of a finite set of predefined constructions, which will ensure the control of data access. All data is returned in the GeoJSON format [48], which is a widely supported encoding for geographic data structures (leaflet [49], Google Maps API [50] and Bing Maps API [51]). Additionally, a website using the *Speeding Visualisation* API is developed, as a showcase.

Implementation 4

This chapter contains the implementation details of the DriveLaB system. It is structured similarly to Chapter 3 - *Analysis & Design*, except that the layers contain application components providing functionality, instead of just functionality terms. To further illustrate this, compare Figure 3.1 to Figure 4.1 below.

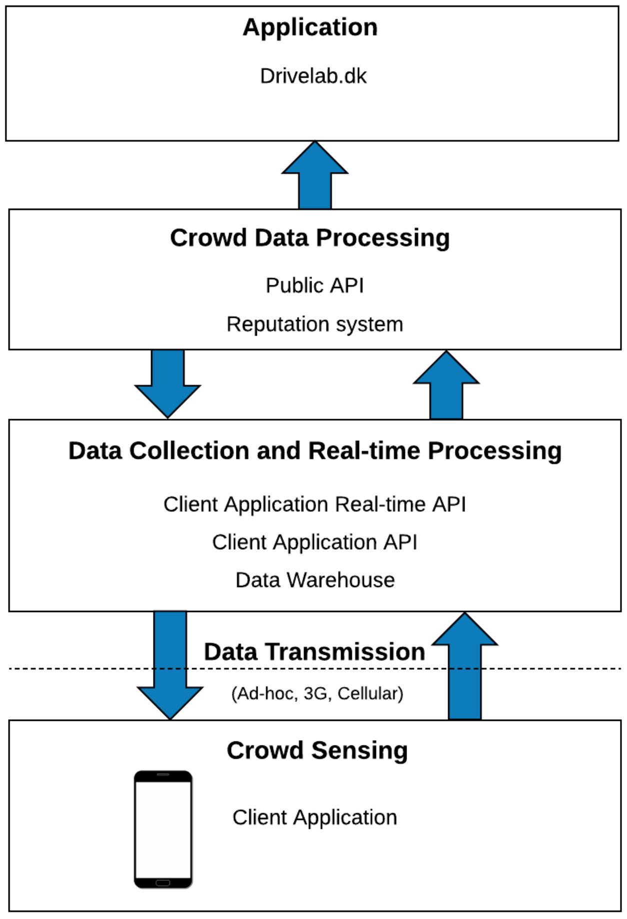


Figure 4.1: MCS DriveLaB Architecture

Eight different applications are running as part of the DriveLaB system, spread across the layers of the MCS framework. Starting from the bottom, the *Client Application* resides in the *Crowd Sensing* layer which communicates through the *Data Transmission* layer to two out of three applications in the *Data Collection and Real-time Processing* layer, namely the *Client Application Real-Time API* using Websockets and the *Client Application API* using

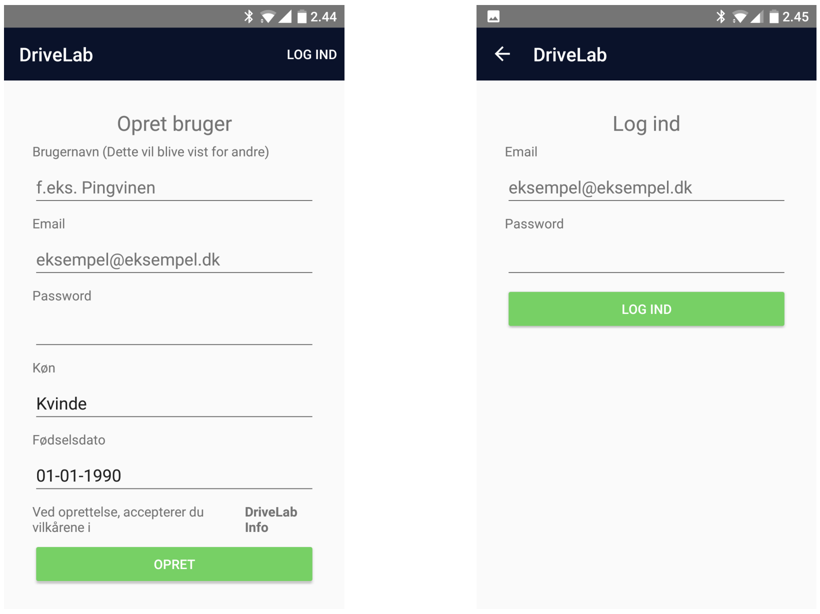
REST. These applications utilise the *Data Warehouse* in the same layer. There are two applications in the *Crowd data processing* layer: *Reputation application* which process data from the data warehouse and stores back the results, and *Public API* exposing anonymised data to the *Application* layer. In this layer, the example application `www.drivelab.dk` presents data collected in the system to the public eye.

4.1 Crowd Sensing

The Crowd Sensing layer contains the client application, which is presented in this section. The presentation includes screenshots of the application, an explanation of the Xamarin project structure, and an exposition of the application architecture.

4.1.1 Client Application

The client application available for Android and iOS is presented in this section, including screen dumps, and a few selected implementation sections containing details of solutions to essential problems. The application is implemented largely based on the assumption that an internet connection is always available even though that is a naive assumption. The only functionality implemented with measures to handle internet connection loss is the tracking and websocket server connection, as these parts are vital to avoid data corruption. Supporting loss of internet connection in all aspects of the system would be a significant time-consuming task and increase code complexity but would, on the other hand, enhance the user experience.



Opret bruger

Brugernavn (Dette vil blive vist for andre)

f.eks. Pingvinen

Email

eksempel@eksempel.dk

Password

Køn

Kvinde

Fødselsdato

01-01-1990

Ved oprettelse, accepterer du vilkårene i **DriveLab Info**

Log ind

Email

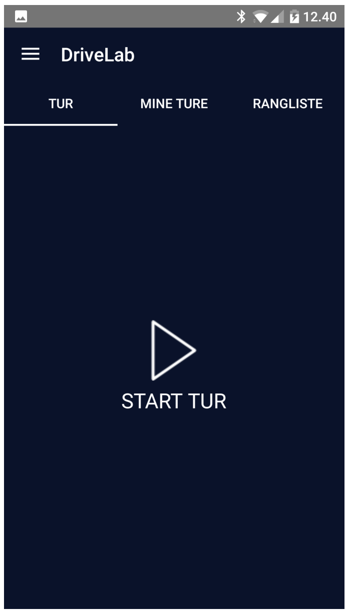
eksempel@eksempel.dk

Password

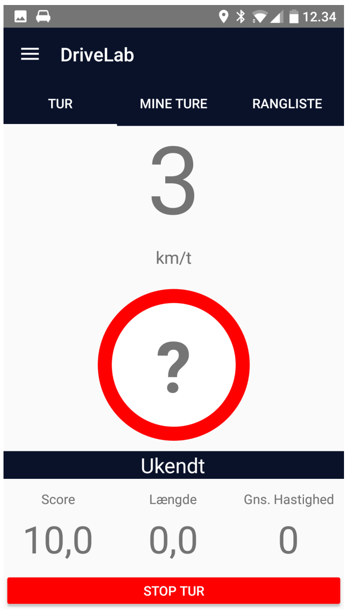
(a) Sign Up Page (b) Login Page

Figure 4.2: Access Pages

The application is available for download today on Google Play Store for Android and App Store for iPhone. The Figure 4.2a is opened upon launching the application, if not already logged in. The user can create an account by providing a username, email, password, gender and lastly the date of birth. When signed up, the user has acknowledged the terms of service, as known from many other applications requiring account creation. There is a button in the upper right corner of this page which navigates to Figure 4.2b. This page is for already registered users, enabling account migration across smartphones, which were not possible for the application presented in [3].



(a) Tracking Inactive

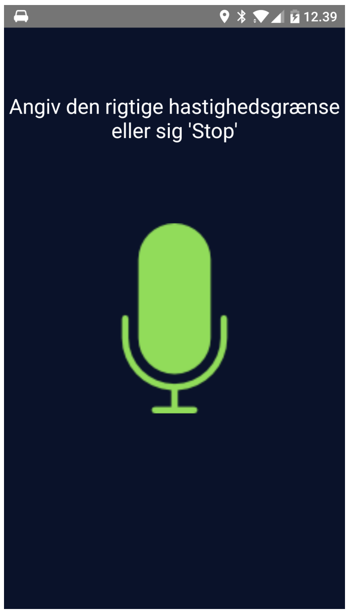


(b) Tracking Active

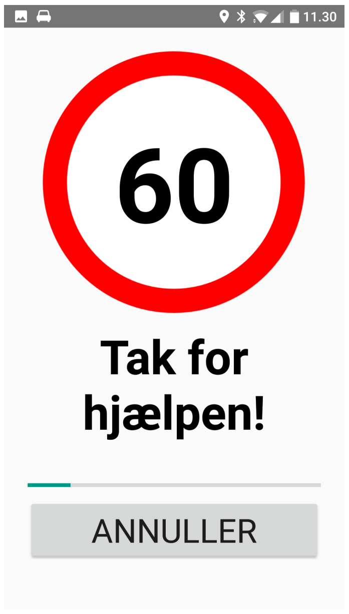
Figure 4.3: Trip Page (Landing Page)

After either creating an account or logging in, the user is presented to Figure 4.3a which presents the options to open the menu at the upper right corner button with the conventional menu icon. Further, the user can utilise the menu tabs to navigate to their trip history or the leaderboard. Lastly, the page offers to start a trip by pressing the button or text in the middle of the screen. This will navigate to Figure 4.3b and start the tracking using a foreground service, as denoted by the car icon in the status bar on top. This service is application independent in the sense that closing the visual application will not affect the service and tracking will continue. Similarly, the service is spawned when using automatic tracking without opening the actual application.

The tracking page in Figure 4.3b displays real-time driving information with the driving speed, road speed limit, road name, score, trip length and average speed. A red button at the bottom presents the ability to stop the trip when arriving at the destination. A hint is shown on top of the speed and speed limit when the page is loaded, which states that the covered area can be pressed to report a faulty speed limit.



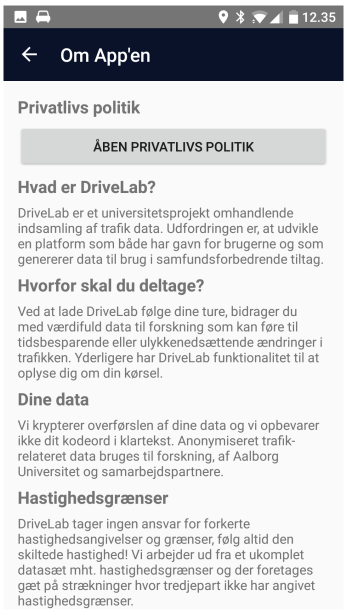
(a) Speed Limit Speech Recognition Page



(b) Cancel Timer Page

Figure 4.4: Speed Limit Report Pages

Falsely assigned speed limits can be reported by navigating to Figure 4.4a which uses speech recognition. The large microphone icon depicted on this screen is white when the device is not listening and green as shown when the device is ready to listen to user speech. The implementation details on parsing speed limits from user speech are further elaborated in Section 4.1.1 - *Crowdsourcing Speed Limits*. If the speech recognition parses the word “stop”, the reporting is aborted. If it instead parses a valid Danish speed limit, Figure 4.4b is presented with the parsed speed limit, gratitude for contributing text and a button to cancel the report. A timer set to 5 seconds is visualised by a progress bar and indicates the time available to click cancel. Otherwise, the report is submitted.


12.35

Om App'en

Privatlivs politik

ÅBEN PRIVATLIVS POLITIK

Hvad er DriveLab?

DriveLab er et universitetsprojekt omhandlende indsamling af trafik data. Udfordringen er, at udvikle en platform som både har gavn for brugerne og som genererer data til brug i samfundsforbedrende tiltag.

Hvorfor skal du deltage?

Ved at lade DriveLab følge dine ture, bidrager du med værdifuld data til forskning som kan føre til tidsbesparende eller ulykkenedsættende ændringer i trafikken. Yderligere har DriveLab funktionalitet til at oplyse dig om din kørsel.

Dine data

Vi krypterer overførslen af dine data og vi opbevarer ikke dit kodeord i klartekst. Anonymiseret trafik-relateret data bruges til forskning, af Aalborg Universitet og samarbejdspartnere.

Hastighedsgrænser

DriveLab tager ingen ansvar for forkerte hastighedsangivelser og grænser, følg altid den skiltede hastighed! Vi arbejder ud fra et ukomplet datasæt mht. hastighedsgrænser og der foretages gæt på strækninger hvor tredjepart ikke har angivet hastighedsgrænser.

Figure 4.5: About Page

An about page has been implemented for people to seek information about the project and what they contribute to by participating. Likewise, Android requires apps that use privacy-sensitive permissions, like locations and audio, to have a *Privacy Policy* available. This policy contains an explanation of which permissions the app uses and a reasoning for using these permissions. A direct link (`http://drivelab.dk/privacy_policy.php`) to the privacy policy associated with this application is implemented on top of the about page. Like in [3], this application also contains pages for trip history, leaderboard, automatic tracking settings and sound settings. These serve mostly the same purpose as in [3], and a description is therefore omitted in this report.

Xamarin Application Architecture

The primary advantage of using Xamarin for cross-platform development is sharing code in the business layer across platforms. However, Xamarin Forms [52] makes code sharing of user interface possible too, contingent on using universal interface features. The DriveLaB application is created using Xamarin Forms with a *shared project* [53] containing the shared code. A shared project is a project without any output; rather the project is copied into projects referencing them. In this case, the Android and iOS projects as seen on Figure 4.6.

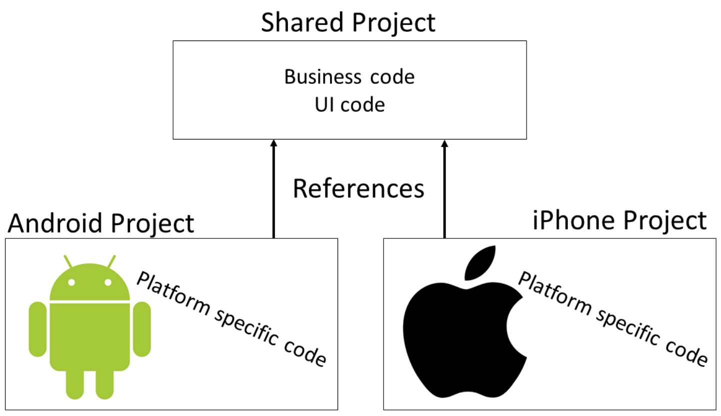


Figure 4.6: DriveLaB Xamarin Project Architecture

The *platform specific code* in each platform project is code related to the following parts of the application:

- *Locations* - Extracting locations from the device GPS
- *Audio* – Extracting device volume and playing sounds
- *Speech Recognition* – Enabling speech recognition and parsing of speed limits
- *Bluetooth* – Extracting paired Bluetooth devices and subscribing to connections (Android only)
- *Lifecycle Events* – Utilising services (Android only)

These features equivalents roughly into 350 lines of code which are platform specific. That is a relatively small amount of code and represents only a fraction of the entire code in the client application, which is a result of efforts towards achieving as much code sharing as possible.

Xamarin maintains a GitHub repository [54] of officially supported plugins which enable further code sharing. However, the maturity and support of these plugins vary in quality and the group made several efforts towards improving the project relevant plugins. Especially the *GeoLocator* plugin [55] received issue descriptions about location precisions [56], new beta package errors [57] and plugin bugs when used on a Huawei smartphone [58], which led to a plugin update. The location related features are yet implemented platform specific, despite the efforts to contribute and gain more code sharing, as the plugin did not reach an acceptable state regarding the precision of locations.

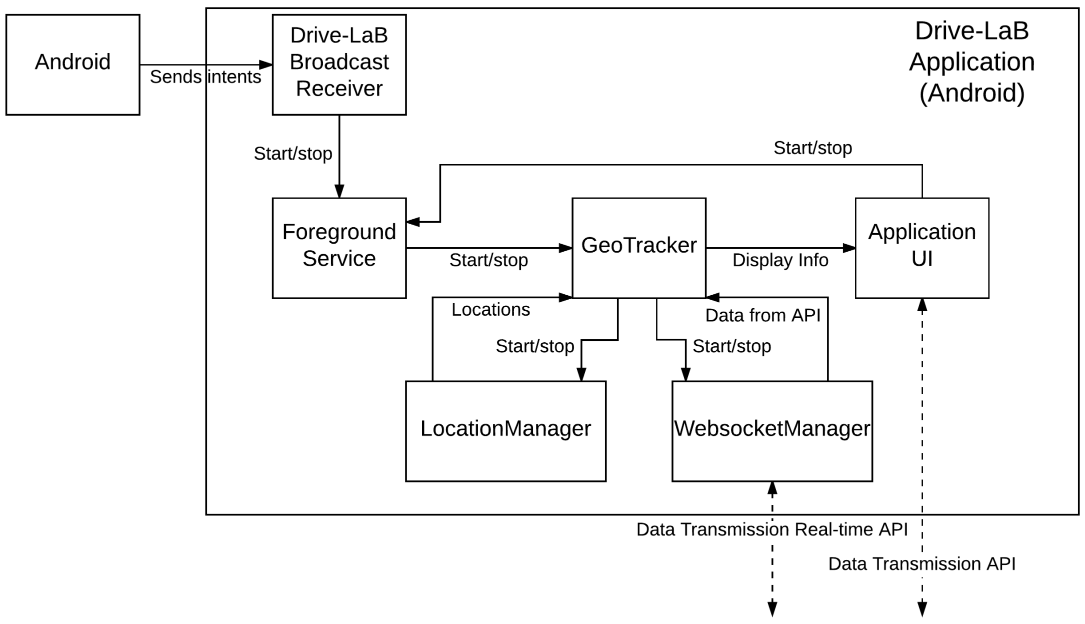


Figure 4.7: DriveLaB Android Internal Architecture

Figure 4.7 shows the internal component architecture of the DriveLaB Android version. The boxes represent components, and the arrows represent the main purpose or usage. The *Android* depicted in the upper left corner acts as the Android OS component responsible for sending intents, in this case to the *DriveLaB Broadcast Receiver.* The receiver can start and stop the *Foreground Service* as later elaborated in details in Section 4.1.1 - *Automatic Tracking.* The *GeoTracker* component is implemented using the Singleton pattern [59] to ensure object sharing between the service and *Application UI.* The service can toggle tracking in the GeoTracker, and the UI displays the real-time information available: speed, speed limit, road name, score, trip distance and average speed. The UI can start and stop the service similar to the broadcast receiver, thus enabling manual tracking. The UI also communicates with the Section 4.3.2 - *Client Application API* through the data transmission layer.

As mentioned and illustrated, the GeoTracker handles extracting locations from the *LocationManager* and sends these and the user's speed limit corrections through the *Real-time API.* The API returns the information which is then drawn to the UI display.

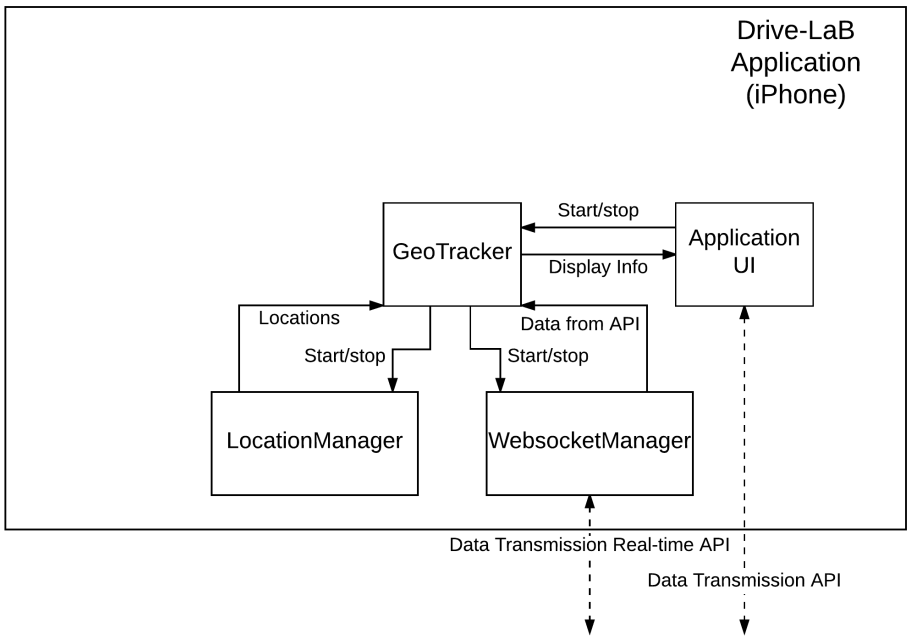


Figure 4.8: DriveLaB iOS Internal Architecture

Figure 4.8 bears significantly resembling in the components common to Android. However, iOS does not expose similar functionality as *Android* in Figure 4.7, which is the current obstacle to supply iOS with a similar solution of automatic tracking. This is because there are no equivalent for a broadcast receiver in iOS. The iPhone Application UI can directly toggle the tracking in the GeoTracker and thus, enables manual tracking. Otherwise, the GeoTracker and WebsocketManager is identical to Android due to the shared project code.

Automatic Tracking

The utilities to facilitate automatic tracking on Android is specified in Figure 4.7 as a *Broadcast Receiver* (BR) and a *Foreground Service* (FS). The BR provide the functionality to detect Bluetooth devices connecting and disconnecting to the smartphone, whereas the FS is a service that facilitates the continuous tracking in the background of the smartphone. As the solution to automatic tracking only applies to Android, the code for both components resides in the Xamarin Android project. Listing 4.1 presents the implementation of the BR.

```
 1 [BroadcastReceiver(Enabled = true)]
 2 [IntentFilter(new[] { "android.bluetooth.device.action.ACL_CONNECTED",
       "android.bluetooth.device.action.ACL_DISCONNECTED" })]
 3 public class DriveLabBluetoothReceiver : BroadcastReceiver
 4 {
 5     public override void OnReceive(Context context, Intent intent)
 6     {
 7         BluetoothDevice device = ExtractDeviceFromIntent(intent);
 8 
 9         if (device.Address != Settings.AutoTrackDeviceMacAddress)
10             return;
11 
12         Intent serviceIntent = new Intent(context, typeof(DriveLabService));
13 
14         if (intent.Action == "android.bluetooth.device.action.ACL_CONNECTED")
15         {
16             serviceIntent.SetAction("start_foreground");
17             serviceIntent.PutExtra("device", device);
18             context.StartService(serviceIntent);
19         }
20         else if (intent.Action == "android.bluetooth.device.action.ACL_DISCONNECTED")
21         {
22            context.StopService(serviceIntent);
23         }
24     }
25 }
```

Code Snippet 4.1: Bluetooth Connectivity Broadcast Receiver

The code lines are explained in chronological order:

- *Line 1* – The BroadcastReceiver attribute is a feature in Xamarin to specify the broadcast receiver in the Android Manifest [60] automatically.
- *Line 2* - The IntentFilter attribute adds to the Android Manifest the intents that the broadcast receiver should receive. The ACL_CONNECTED and ACL_DISCONNECTED intents denote a Bluetooth device connection and disconnection, respectively.
- *Line 3* - The DriveLabBluetoothReceiver class is defined and inherits from BroadcastReceiver to enable registering the class as a broadcast receiver.
- *Line 5* - The inherited OnReceive method is overridden, which is the method executed upon receiving an intent.
- *Line 7* - Extracts the Bluetooth device object from the intent. The object contains the name, type, UUID, etc.
- *Line 9* - This comparison is the *stored device?* decision depicted on Figure 3.2 - *New Automatic Tracking Flowchart*. The AutoTrackDeviceMacAdress holds the user chosen paired Bluetooth device MAC address.
- *Line 12* – A new intent is instantiated, used to either start or stop the service
- *Line 14* – A comparison of the intent's action property reveals whether a Bluetooth device connected. If so, lines 16-18 set an action and attach the Bluetooth device to the intent, which is used to start the service.

- *Line 20* – Contrary, if the intent's Action indicates a device disconnection, line 22 stops the service. The service can close the tracking mechanism properly upon being shut down.

The intents are received reliably but occasionally with a slight delay (<2 sec). There are found one issue in the implementation that is addressed further in Section 5.1.1 - *Automatic Tracking*.

Crowdsourcing Speed Limits

Knowledge of speed limits is a significant part of the DriveLaB system, and the absence thereof poses a threat against user experience and system viability. The creation of functionality to support user contribution is carefully considered to be as little distracting as possible due to the context of driving. As noted in Section 3.1 - *Mobile Crowd Sensing (Client Application)* any physical phone interaction is illegal, so the project group attempted to implement continuous speech recognition. There are several obstacles as to why this solution did not succeed:

- *Android User Experience* – Implementing continuous speech recognition on Android resulted in significant gaps in registering for user's voice and sounds would play to indicate both starts and stops listening often. Measures to avoid the sounds were not accordance with Android design guidelines.
- *Apple Limits* – Apple offers their speech recognition capabilities free of charge, however continuous use of this results in Apple requiring payment due to excessive use.
- *Battery Drain* – Listening to voice continuously involve constant use of microphone and processing power which drains the battery substantially.
- *Data consumption* – The data transfer would increase a considerable amount as every microphone input is sent to either Google or Apple for speech recognition.

The selected solution consists of a physical touch on the majority of the tracking page, which then activates speech recognition in a short period. The physical interaction is comparable to a physical interaction with the car stereo and is intended to be possible without distraction from the driving task.

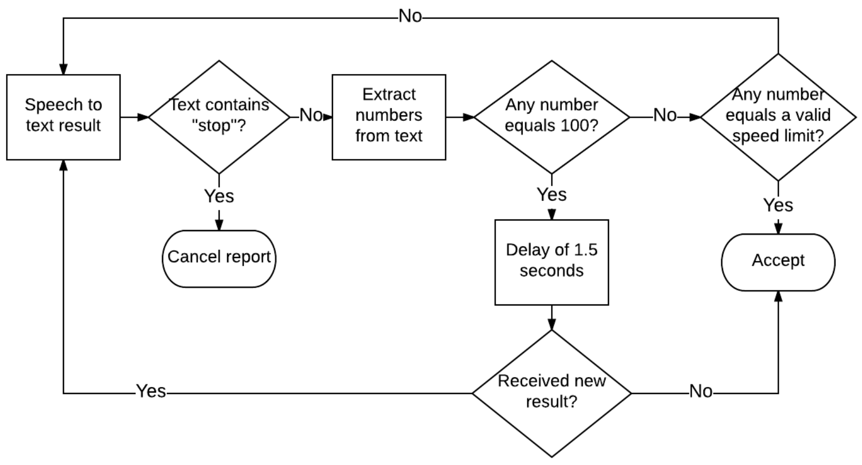


Figure 4.9: Speed Limit Parsing Logic Flowchart

The speech recognition APIs on Android and iOS is continually evaluating the speech input, and the resulting partial string is equally often inspected for valid speed limits. The approach of parsing partial results increases user experience as the application promptly reacts to legal speed limits instead of waiting a fixed amount of time before parsing the full text.

Figure 4.9 depicts the logic to parse the partial results returned by the API and starts at the *Speech to text result* state. At first, the text is searched for the word “stop” as this enables the user to cancel the reporting. If it is not present, all numbers are extracted from the text which prepares for the detection of the number 100. This number represents a particular case when utilising speech recognition in partial result mode. In Danish and English, speaking the number 130 is pronounced “one hundred and thirty”, which would lead to the speech recognition parsing the speed limit report incorrectly as 100. A *delay of 1.5 seconds* counters this issue. By delaying the acceptance of 100, the text “130” retrieved next from speech recognition has enough time to be parsed and accepted as the correct speed limit.

Reported speed limits goes through the reputation system for validation as described in Section 4.4.2 - Reputation System

4.2 Bidirectional Real-time Data Transmission

Data Transfer Objects (DTOs), used for the communication between the client and API, are defined in a shared project and referenced by both the client and API project transpired to be a beneficial development oriented advantage by providing a common structure. Altering a DTO to support a new feature on the client application reveals the impact on the API upon next build of the code, which reduced bugs and heightened the development speed. The DTOs and their purpose in this project are:

- *ErrorDto* – Contains an error code and message. The API sends an ErrorDto to the client as a response to an erroneous REST API request.

- *LeaderboardEntryDto* – Contains the name and total score of an active user on the leaderboard.
- *GpsPointDto* – Contains the location data received from the smartphone GPS.
- *LocationInfoDto* – Contains a trip id and a list of GpsPointDtos. The clients send the DTO to the API in a fixed interval while tracking.
- *LoginDto* – The client sends a LoginDto with credentials upon attempting to log in.
- *RoadDto* – The API sends a RoadDto containing the street name and speed limit to users that are tracking.
- *SpeedLimitReportDto* – This DTO is sent from the client to the API when a correction of speed limit has been reported.
- *StartInfoDto* – The user initiates a trip on the API by sending a StartInfoDto, which contains the details of the Bluetooth device if the tracking was initiated automatically.
- *TotalScoreDto* – The total score of a user is presented in the side menu of the client application. This is requested upon application launch and trip stop, and arrives in a TotalScoreDto
- *TripDto* – Contains full trip information and is used to display the users trip history.
- *TripStatusDto* – The TripStatusDto is sent from the API to the client continuously while the user is tracking. It includes real-time calculated score, trip distance and average speed.
- *UserCreateDto* – A UserCreateDto is sent to the API when the user signs up in the client application and includes the username, hashed password, gender, email, and date of birth.

The DTOs are either used in the request or response of the REST API or as part of a websocket message type previously presented in Section 3.2 - *Data Transmission.* The real-time message types are introduced in Table 4.1 with their *id*, *name*, *purpose*, and *DTOs* included. The DTOs from the above list which are not included in any of the message types are utilised in the REST API.

Id	**Name**	**Purpose**	**DTOs included**
1	StartTrip	Client initiates a trip to the server	StartTripDto
2	Locations	Client sends a batch of locations to the server	LocationInfoDto, GpsPointDto
3	Received	Server response to a websocket message from the client	None
4	SpeedLimitReport	Client speed limit correction report	SpeedLimitReportDto
5	RoadInfo	Server sends the road information of which the user is currently drives on	RoadDto
6	TripStatus	Server sends trip information to the user	TripStatusDto
7	UserFeedback	Server sends an acknowledgement message of good driving to the user	None
9	StopTrip	Client signals the server to stop the trip	None

Table 4.1: Real-time Message Types

The message types *Received*, *UserFeedback*, and *StopTrip* includes no DTOs. *Received* and *StopTrip* act as signals without accompanying data, whereas UserFeedback has the appraisal text string included, which is so simple that adding a dedicated DTO would increase complexity unnecessarily. All messages are AES encrypted and sent through the websocket protocol as outlined in Figure 3.3.

4.3 Data Collection and Real-time Processing

As described in Section 3.3 - *Data Collection and Real-Time Processing*, this layer has the purpose of retrieving, processing and storing data contributed by the users during their trips. Additionally, mechanisms are implemented to filter away low-quality data and provide incentives for users to participate.

The component *Client Application Real-time API* has the task of receiving GPS data, the audio configuration of users' smartphones, and speed limit reports with the use of the *WebSocket* protocol as described in Section 4.2 - *Bidirectional Real-time Data Transmission*. Additionally, GPS data is processed to provide real-time driving related information to users during their trips and stored in the data warehouse afterwards.

Client Application API is a RESTful API handling the creation, login, and validation of users. It also provides information and details about the users' previous trips and a leaderboard that displays the performance of active users.

ASP.NET Core[61] was used to develop these components in the last project, and it is decided to preserve this choice of technology since the project group has had mainly positive experiences with it and saw no reason to change. The real-time API is rewritten from scratch due to the abandonment of SignalR, described in Section 3.2 - *Data Transmission*, and the need for implementation of new features which require architectural changes. Additionally, the *Object-relational mapper* (O/RM), Entity Framework Core (EF Core)[62], is used to handle database interaction, which also influences the decision to build a new real-time API application.

The *Data Warehouse* handles storage of all data and trip processing which enables the users' routes to be displayed through the *Public API* described in Section 4.4.1 - *Public API* in addition to validation of users in the creation process.

4.3.1 Client Application Real-time API

This section presents the *Client Application Real-time API* component where it will be described how it handles the trips and communication with the *Client Application* described in Section 4.1.1 - *Client Application*.

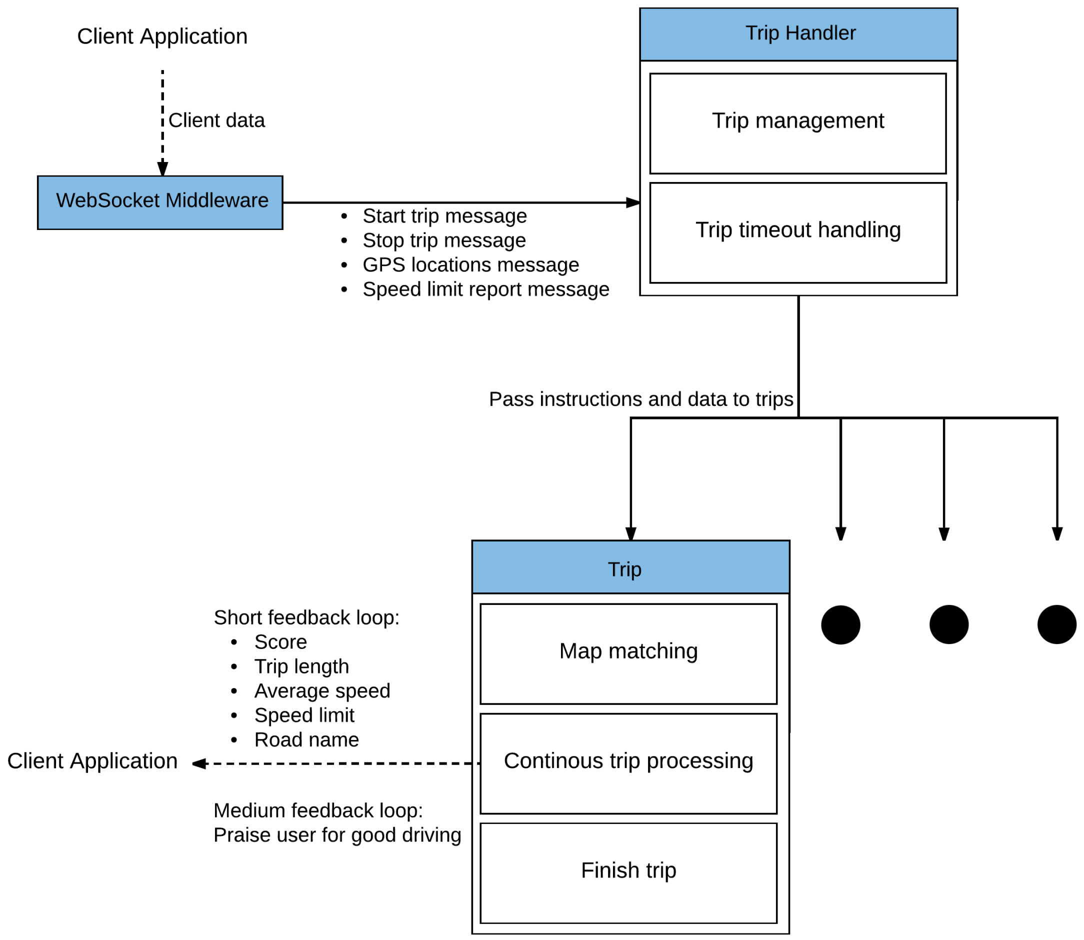


Figure 4.10: Real-time API Architecture

In Figure 4.10, the text in the blue boxes are objects, and the text in the white boxes are tasks related to those objects. The arrows indicate the data flow between them. The *WebSocket Middleware* handles data received from the *Client Applications* where it decrypts and deserializes the JSON strings as described in Section 4.2 - *Bidirectional Real-time Data Transmission.*

Trip Management

The messages are passed on to the *Trip Handler*, which has the job of managing the trips' lifecycle. Depending on the message passed by the *Client Applications*, the *Trip Handler* either creates a new trip, request an ongoing trip to finish and thereafter removes it or passes a speed limit report or GPS locations to a trip. A dictionary is used to store the *Trips* with the GUID as the entry key. This GUID is generated upon creation for each *Trip* by the *Client Application.* The risk of two identical GUIDs being generated is astronomically low. However, this scenario is currently not handled and could cause an exception to occur if identical GUIDs are generated for users within the same timeframe of driving.

Trip Timeout Handling

During a trip, the websocket connection for some trips might end abruptly due to loss of internet connection. The *Client Application* will therefore not send a stop signal, which leaves the server responsible for making sure that the trip gets finished. This is accomplished by marking trips with a timestamp of when they last received GPS locations. The server will have an event which triggers every five minutes where the timestamp of each trip is checked. Trips that have not received any GPS locations within five minutes are then finished and removed from the dictionary. If a trip reconnects within five minutes, the new websocket connection will be assigned to the corresponding *Trip* object, which allows the tracking to go on.

Map Matching

As mentioned, the real-time map matching algorithm outlined in Chapter 2 - *Map Matching* is implemented. The algorithm consists of the three states, *Initial MM*, *MM on a link*, and *MM at a junction*, as explained earlier. Both *Initial MM* and *MM at a junction* collects a list of candidate roads by drawing a circle with the GPS point accuracy as the radius. This is done by using the two PostGIS functions `ST_Buffer` to construct a border box and `ST_Intersects`, finding roads intersecting the border box. The approach is performed for each GPS point processed in the two modes *Initial MM* and *MM at a junction*, including a network overhead to the database each time.

All three modes project the GPS point on to a road. Even though, orthogonal projection is achievable in PostGIS by utilising the two functions `ST_Line_Interpolate_Point` and `ST_Line_Locate_Point`, it is decided to embed such feature in the *client application real-time API.* Embedding the projection feature reduces the database interaction, which should lead to better performance.

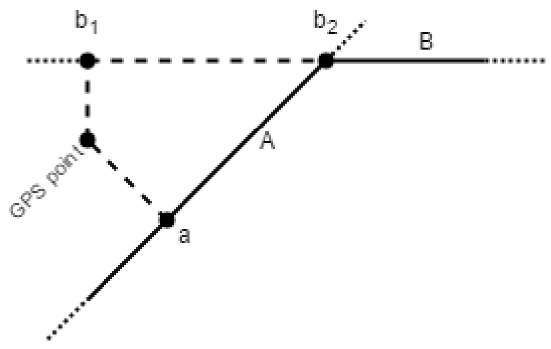


Figure 4.11: GPS Point Projection

Figure 4.11 - *GPS Point Projection* visualises the two possible projection scenarios. Line A and B both represent roads where the fine dotted lines illustrate that roads are treated as infinite lines. Point a, b_1, and b_2 illustrate the projections used in the two scenarios elaborated in the following list.

- *Scenario A* – The GPS point can be directly projected to A such that, the line from the GPS point to point a is perpendicular to the road line A.
- *Scenario B* – The GPS point cannot be directly projected to B. However, a point b_1 can be found such that, a line between the GPS point and b_1 is perpendicular to the infinite version of B. Finally, the endpoint of B, with the lowest distance to b_1 will represent the projected point b_2.

The distance from the GPS point to the road is in *Scenario A* defined as $\|GPSpoint - a\|$ and in *Scenario B* it is $\|GPSpoint - b_1\| + \|b_1 - b_2\|$. The Euclidean distance is used due to its arithmetic simplicity, even though it does not account for the curvature of the earth. The curvature is ignored as GPS points are often accurate within a few metres, which should minimise the overall relevance.

Continuous Trip Processing

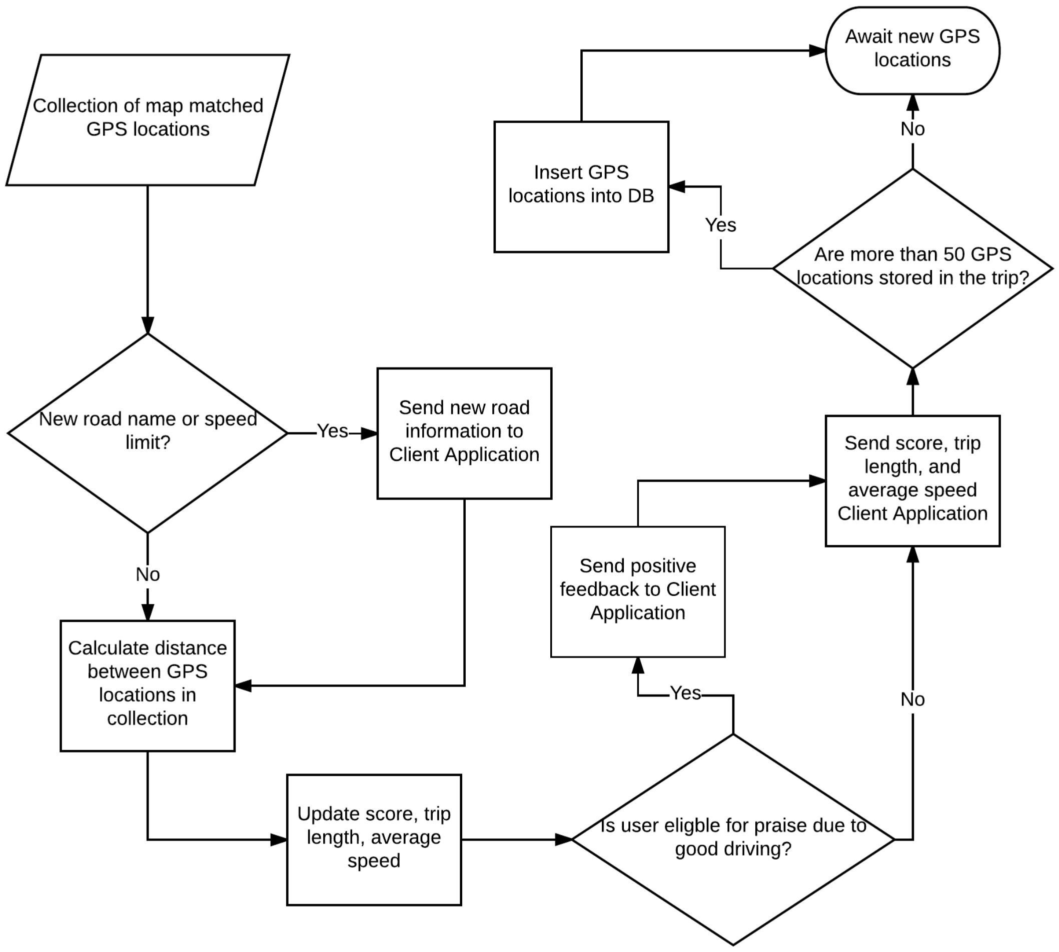

Collection of map matched GPS locations
New road name or speed limit?
Yes
Send new road information to Client Application
No
Calculate distance between GPS locations in collection
Update score, trip length, average speed
Is user eligble for praise due to good driving?
Yes
Send positive feedback to Client Application
No
Send score, trip length, and average speed Client Application
Are more than 50 GPS locations stored in the trip?
Yes
Insert GPS locations into DB
No
Await new GPS locations

Figure 4.12: Continuous Trip Processing

As seen in Figure 4.12, once a batch of GPS locations has been map matched, the latest GPS location is used to determine whether the user is driving on a new road, or the speed limit has changed. If that is the case, then the information is transmitted to the *Client Application*.

Hereafter, the beeline distance between the GPS locations is calculated using the *Haversine formula* [63] as it takes the earth's spherical shape into account. The implementation of the formula made by Veness [64] is adopted for better performance.

The distances between the GPS locations are used to update the performance score, which is calculated in the same manner as in [3]. It is the ratio between the total distance driven and the punishment value received by driving above the speed limit, which determines the score. The punishment value is based on the distance driven above the speed limit in addition to how much the speed limit is exceeded in percentage. The more a user exceeds the speed limit, the higher punishment value is added.

The medium feedback loop, described in Section 3.3.1 - *Incentive Mechanisms*, has the purpose of providing positive feedback to the user. This is done by praising the user whenever the speed limit has not been exceeded on a continuous road section by a certain number of kilometres. The user can receive praise several times throughout a trip, however, it becomes increasingly harder. Praise is received for the first time after four kilometres, which doubles each time hereafter. If a user is eligible to praise, the string *"You have driven X kilometres under the speed limit, well done!"* is transmitted to the *Client Application* and played through audio with the use of text to speech.

The updated performance score, trip length, and average speed, which are all part of the short feedback loop, gets transmitted to the *Client Application* after the processing of every GPS location batch. The trip temporarily stores the processed GPS locations and inserts them into the database when 50 have accumulated.

Finish trip

Once the trip is over, the *Client application* transmit a stop signal. The overall statistics of the trip are updated, such as performance score, start/stop location, start/stop time, weather information, and trip length. As mentioned in Section 3.3.3 - *Data Quality Maintenance*, the validity of the trip is determined by considering its length. If it is less than 500 metres, it is removed from the database. This is done to ensure that trips started by accident are removed to avoid storing polluted data.

4.3.2 Client Application API

The *client application API* has many similarities to the REST API presented in [3]. Firstly, the URL and port number is unchanged and is therefore still *http://stream.cs.aau.dk:9220*. Secondly, similarly to the REST API presented in [3], the API is accessed by appending */api* to the URL. Table 4.2 gives an overview of the API calls possible to perform.

API path	Method	Input	Response
/user/create	POST	UserCreateDto	UserCreateDto
/user/login	POST	LoginDto	LoginDto
/user/totalScore/guid/#1	GET	#1 - GUDI (string)	TotalScoreDto
/trip/guid/#1/start/#2/amount/#3	GET	#1 - GUID (string) #2 - List offset (integer) #3 - Limit (integer)	TripDto list
/trip/latest/guid/#1	GET	#1 - GUID (string)	TripDto
/leaderboard/rollingwindow	GET	-	LeaderboardEntryDto

Table 4.2: API Layout

An example of utilising one of the API calls presented in Table 4.2 could be `http://stream.cs.aau.dk:9220/api/leaderboard/rollingwindow`. Three out of the six presented URLs requires a GUID before testing the API call is possible. Therefore, a working GUID *f730c8ca-0c95-4aa3-9917-4e6dbd5d2de0* is made available.

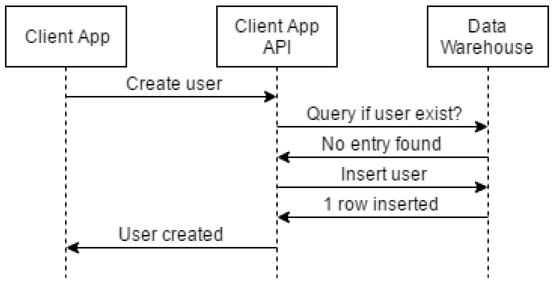
Versioning

As mentioned in Section 3.4.2 - *Versioning* it is decided to use custom header fields for REST API version control. The implementation makes use of the *Microsoft.AspNetCore-.Mvc.Versioning* NuGet package provided by Microsoft. The package is set to recognise an *Api-Version* header field, accepting versions in the format *major.minor* (e.g. 0.1). If no version header is set, the API will default to a prespecified version. However, it is encouraged to always specify the version header to avoid compatibility issues should the prespecified version number be increased.

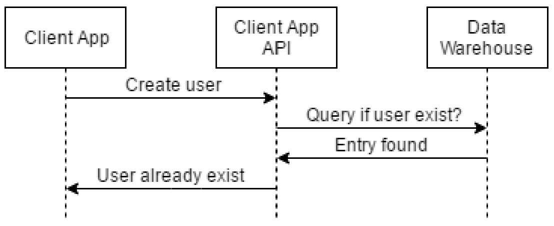
4.3.3 Data Warehouse

As mentioned, weather data is deemed valuable in a UBI context to assess both GPS quality and responsible driving. To collect the weather data *OpenWeatherMap API* [65] is used, which offers 60 API calls for free each minute. Furthermore, the free subscription plan only guarantees the weather data update interval to be less than 2 hours. This signifies that only every second hour, one can be sure to retrieve newly measured data. Therefore, due to the low update rate of the weather data, duplicates would often be stored. Additionally, since every user performs insertions in the *gps_fact* table each second, 60 API calls will not be sufficient. The assumption in this project is that significant weather changes will not normally take place within a radius of 50 km. User's between 30 and 59 years of age are the most frequent travellers, and their travel distance is 34.9 km on average each day [66]. It is in this project believed that associating weather data for a whole trip is sufficient. At the current state, the system does not utilise the collected weather data; however, it is possible to incorporate in future feature implementations. As the data is already collected and associated with trips, new weather-related features may function retroactively.

For the new user registration feature to be useful, some level of authentication is required. Figure 4.13 and Figure 4.14 are two different approaches to prevent user account duplications.



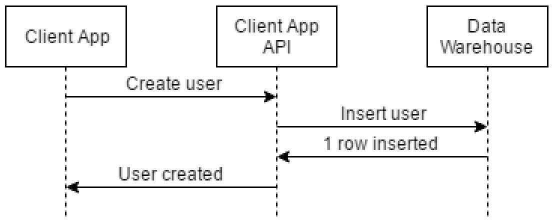
(a) Successful User Creation



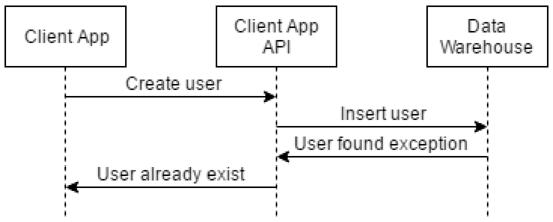
(b) Unsuccessful User Creation

Figure 4.13: User Authentication Approach with Transaction

The transaction-based approach presented in Figure 4.13, requires two separate database requests in the worst case. As the client application API and the database runs on two separate servers, a network overhead for each request is included.



(a) Successful User Creation



(b) Unsuccessful User Creation

Figure 4.14: User Authentication Approach with Database Trigger

The database trigger approach presented in Figure 4.14, however, requires one request at most. This is possible by triggering a database routine before any insert statement is performed on the *user_ dim* table. The database routine checks if an email, username, or GUID already exists, and throws an exception if it is the case. Different exceptions are raised depending on which duplicate is in question. Hereby, it is possible to provide information about what went wrong. The logic depicted in Figure 4.14 is the one implemented in this project, because of its simplicity and the limited use of network communication.

A database trigger is also used to populate the *route_ dim* table. A routine is called whenever a row in the *trip_fact* table is updated. It starts by deleting all instance of road segments associated to the specific trip id. If the *processed* column is `True`, it will repopulate the *route_ dim* table with road segments based on an aggregation of associated GPS points found in the *gps_fact* table. If the *processed* column is `False`, however, the routine will simply return, leaving the *route_ dim* table unchanged.

The leaderboard is implemented as a view table, which bypasses the need to schedule updates for a persistent leaderboard table. However, this implementation may not scale well with a fast-growing user base, it is a simple implementation which does perform well enough for the small experiment presented in this paper.

The Danish map offered by Geofabrik [67] is now used in combination with the OSM2PO tool [68]. OSM2PO is primarily used to transform *Open Street Map* (OSM) data into routable maps. In this project, OSM2PO is mainly used for splitting road segments in such a way that no road segment goes through and beyond a junction node. Visualising a trip on a graphical map can thereby be done by drawing the visited road segments directly onto the map. Furthermore, the *road_ vertex_ dim* mentioned in Section 3.3.4 - *Dimension Tables*, is autogenerated by the OSM2PO tool, facilitating the topological analysis done in the MM algorithm.

4.4 Crowd Data Processing

The most important implementation details regarding the public API and the reputation system will be covered by this section. It is clarified which API calls are currently available through the public API as well as a quick demonstration of how to apply them. Furthermore, a brief walkthrough of the components that constitute the reputation system and formalisations of the various equations is provided.

4.4.1 Public API

Contrary to the *client application API*, the *public API* is designed to be used by third party institutions. Allowing the institutions to combine search terms, is a high priority as the public API should apply to a variety of traffic related applications. To accommodate this goal, it is decided to utilise the query string available through the URL. The public API runs as a new application, independent of the application API. Therefore, the public API is accessed through the same domain, yet assigned a different port `http://stream.cs.aau.dk:9250/api`.

API path	Method	Query parameter	Input
/heatmap	GET	age	- (integer, integer) integer> integer<
/heatmap	GET	gender	male female
/soundwarning	GET	-	-

Table 4.3: API Layout

Table 4.3 - *API Layout* show the different constructs offered by the public API. To give an example of use, the following URL `http://stream.cs.aau.dk:9250/api/heatmap?age=(18,25)&gender=male` returns all data available concerning males in the range of 18 to 25 years. As mentioned in Section 3.5 - *Applications* the public API will return all geographically related data in the GeoJSON format to ease utilising the API in third party applications.

4.4.2 Reputation System

The implementation of the reputation system for validating speed limit reports is based on the design considerations described in Section 3.4.1 - *Reputation System.* The programming language *Python3* [69] is used to implement the system. This eases the porting to a database routine written in *PL/Python* [70], which enables scheduling the processing of reports to be conducted at times where the server load is minimal with the use of *pgAgent* [71]. However, due to time constraints, this was not completed.

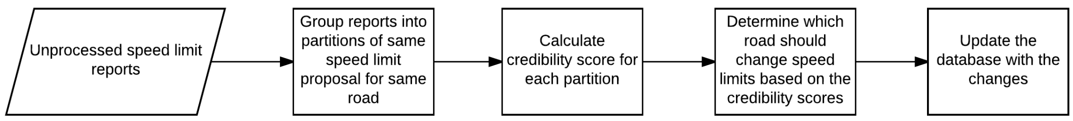

Unprocessed speed limit reports
Group reports into partitions of same speed limit proposal for same road
Calculate credibility score for each partition
Determine which road should change speed limits based on the credibility scores
Update the database with the changes

Figure 4.15: Speed Limit Validation Process

As depicted in Figure 4.15, the unprocessed speed limit reports are organised in such way that reports for the same road segment, and speed limit proposals are grouped together forming a partition. Hereafter, the reputation score is calculated for each partition.

This is done by first calculating a reputation score for each user in each partition.

$$total\_error_u = irp\_within\_60_u * \alpha + irp\_beyond\_60_u * \beta + locs\_outside\_region_u * \gamma \mid \alpha, \beta, \gamma >= 0, \alpha + \beta + \gamma = 1 \quad (4.1)$$

Equation (4.1) shows how the error rate of a user is calculated, where u denotes a user. *irp_within_60* and *irp_beyond_60* denote the incorrect report percentage within and beyond 60 days respectively. The reputation of users at different points in time has different weights because it is believed that the way a user has acted in recent time is a better indication of the quality of future contributions, therefore it should weight more. The number 60, which is roughly two months, is used to separate what is considered recent and past behaviour. This number is chosen because it is assessed that the average user contributes 2 reports a week resulting in a total of eight reports each month. However, eight reports are believed to be too few to estimate the validity of a user, therefore more than one month worth of data is needed. Furthermore, using data from more than two months could punish the user's reputation for too long, if incorrect speed limit reports are submitted. This may be demotivating and make the user stop contributing. If a user is new and has not made any reports yet, the value 0.5 i assigned, as it is unsure whether the user is trustworthy or not.

locs_outside_region denotes the percentage of GPS locations outside the region where the new speed limit is proposed. This is found by placing a boundary box with a radius of one kilometre around the GPS location where the speed limit report is conducted. The percentage of GPS locations outside this boundary box indicate whether the user is local in the area. α, β, and γ are all weight coefficients where their respective values are 0.8, 0.0, and 0.2 in this project.

The weight coefficient β is 0.0 because there is no point in considering the reputation of a user beyond 60 days since the user experiment described in Section 5.2 - *Uncontrolled Experiments* only lasts three weeks. The weight coefficients 0.8 and 0.2 are chosen because of the correctness of a user's previous reports is assumed to be a significantly stronger quality indicator of their future contributions compared to their local area.

OSM road class	**Default (km/h)**	**Speed limit report constraint (km/h)**
41, 51, 63	50	<60
13, 14, 15, 16, 21, 22, 31, 32, 42, 43	80	<100
11, 12	130	≥100 or ≤130

Table 4.4: Deviation from Default Speed Limit

In Table 4.4, the column *OSM road class* shows the grouping of road classes that has the same default speed limit, which has been acquired with the OSM2PO tool, described in Section 4.3.3 - *Data Warehouse*. Column *Default (km/h)* indicate the default speed limit for the corresponding road classes. The last column, *Speed limit report constraint (km/h)*, shows a constraint used to influence the credibility of a partition. If the proposed speed limit does not satisfy the constraint, then the credibility of the speed limit partition is influenced negatively by evaluating the weight *speed_ credibility* to 0.5. If it does satisfy the constraint, it evaluates to 1.0. These values are chosen because reports that do not satisfy these constraints are deemed more unlikely and should have a harder time making an impact. For example, if a group of users reports 70 km/h on a road segment with OSM class 11, then the reputation score for that partition will be influenced negatively as it is classified as a motorway segment with a default speed of 130.

$$partition\_reputation = (1 - \prod_{u \in U} total\_error_u) * \delta + speed\_credibility * \theta \mid \delta, \theta >= 0, \delta + \theta = 1 \tag{4.2}$$

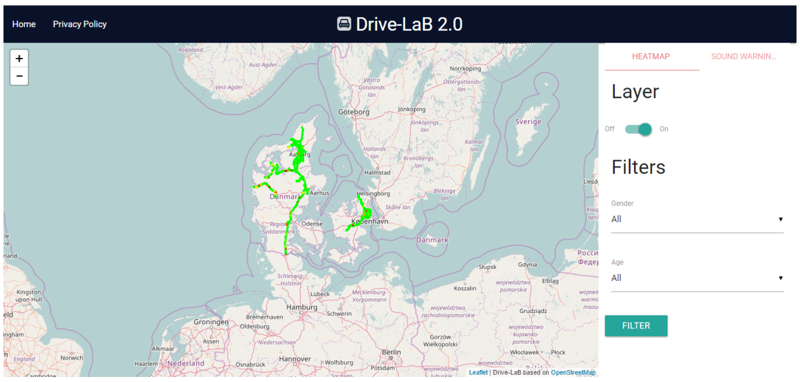
As seen in Equation (4.2), the error rate of all users, where the collection of users within a partition is denoted by *U*, is applied in the calculation of the total credibility score for a partition. Additionally, the *speed_ credibility*, which considers the comparison of default and proposed speed limit is weighted into the calculation. δ and θ are both weight coefficients assigned to 0.7 and 0.3 respectively. The two values for these weight coefficients are chosen because reputable users are assumed to be more trustworthy than the OSM road classification and should therefore weight significantly more in the partition score. This will also help prevent potentially malicious users from reporting a speed limit which differs a lot from the default speed, as their reputation score will be heavily punished by both weights. Once the reputation score has been calculated for each partition, it is determined which speed limits are going to be changed.

$$partition\_reputation > \kappa \tag{4.3}$$

If the *partition_ reputation* has a reputation score that satisfies the threshold denoted by κ in Equation (4.3) and the partition has the highest reputation score of all partitions for the road segment, then the speed limit will be changed. The value for κ is 0.7 in the user experiment, described in Section 5.2 - *Uncontrolled Experiments*. This is chosen because it allows new users to single handily change the speed limit of a road segment which does not deviate a lot from the default speed limit. A slightly lower value for κ would not provide this behaviour. Making it possible for new users to alter speed limits would probably not be wise under normal circumstances. However, since this is a short experiment with relatively few users that are most likely not malicious, it is decided to give them more impact.

4.5 Application

As mentioned in Section 3.5 - *Applications* it is decided to publish a website to showcase the *Speeding Visualisation* CSR opportunity. To ease the process of deploying a website and domain management, the UNOEURO [72] webhotel is used. The website is available through `www.drivelab.dk`.

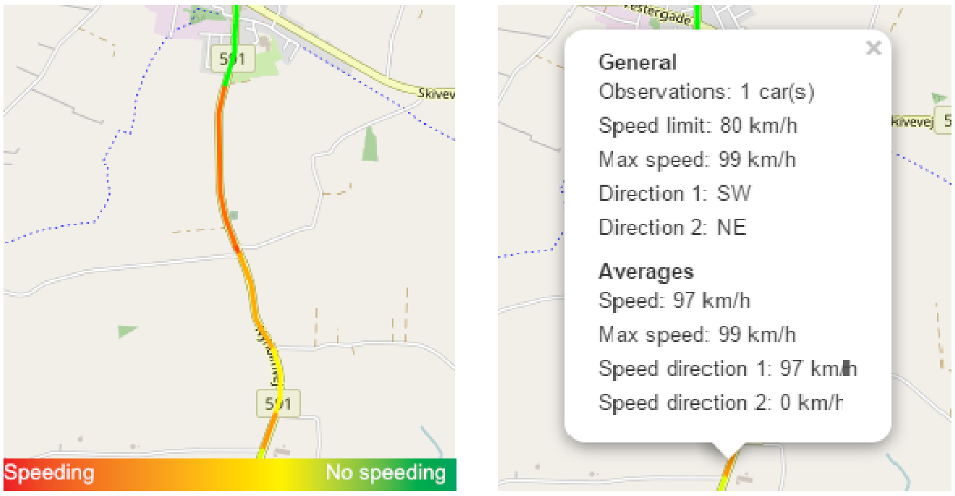


Gender: All
Age: All

Figure 4.16: The DriveLaB Website

4.5.1 DriveLaB.dk

A stack of technologies is used to build the DriveLaB website. PHP is used as the server language; however, no application logic is written in this language. The Leaflet JavaScript library [73] is in conjunction with OSM used to handle the graphical map. It supports GeoJSON by default and therefore able to display the data returned from the *public API* described in Section 4.4.1 - *Public API.* A CSS framework named Materialise [74] is utilised to allow mobile users to navigate the website with ease. Also, the JQuery JavaScript library [75] is used to lighten the *Document Object Model* (DOM) manipulation process.



General
Observations: 1 car(s)
Speed limit: 80 km/h
Max speed: 99 km/h
Direction 1: SW
Direction 2: NE

Averages
Speed: 97 km/h
Max speed: 99 km/h
Speed direction 1: 97 km/h
Speed direction 2: 0 km/h

(a) Aggregated Heat-map (b) Road Information Popup

Figure 4.17: Heat-map Implementation

Figure 4.17a illustrates how a red to green *Hue Saturation Lightness* (HSL) colour scale is used to visualise the degree of average speeding. To limit the HSL colour scale to only red and green, the saturation and lightness are set to 100% and 50% respectively, while adjusting the hue [76] between 0° (red) and 120° (green).

$$HueValue = \begin{cases} 0, & if\ \frac{AverageSpeed - SpeedLimit}{SpeedLimit} \geq 0.3 \\ 120, & if\ \frac{AverageSpeed - SpeedLimit}{SpeedLimit} \leq 0 \\ 120 - \frac{AverageSpeed - SpeedLimit}{SpeedLimit * 400}, & Otherwise \end{cases} \tag{4.4}$$

Equation (4.4) shows the hue value calculation by considering the degree of speeding. The popup screen shown in Figure 4.17b can be triggered by clicking a coloured path, should one require additional information.

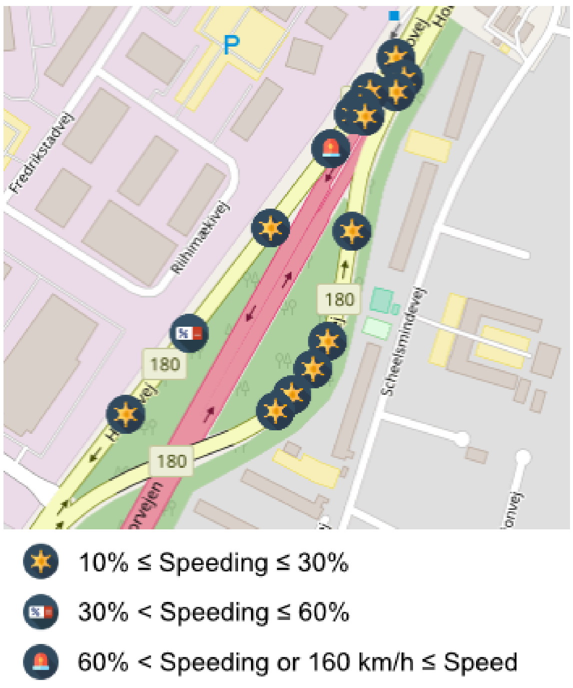


Figure 4.18: Notification Map

Where speeding notifications have been triggered can also be shown on the map as depicted in Figure 4.18. The police batch represents a speeding percentage from 10% to 30%. The white and red ticket indicates speeding higher than 31% and lower or equal to 60% where the siren represents either a speed exceedance over 60% or a speed higher or equal to 160 km/h.

Evaluation 5

This chapter presents controlled and uncontrolled experiments which serve as an evaluation of the implemented system. Controlled experiments are experiments conducted by the authors with a specific evaluation goal. In the uncontrolled experiment, the application is made public for anyone to use where the collected data is analysed.

5.1 Controlled Experiments

In this section, the client application is subject to an experiment regarding the fairness of utilising two platforms, and the automatic tracking solution on Android is evaluated. Additionally, the performance of the real-time API system is tested.

5.1.1 Android and iOS

Fairness is an important aspect of the DriveLaB application. A controlled experiment to investigate the inter-platform fairness was conducted by tracking a trip by both a OnePlus Two Android smartphone and an Apple iPhone 7 smartphone.

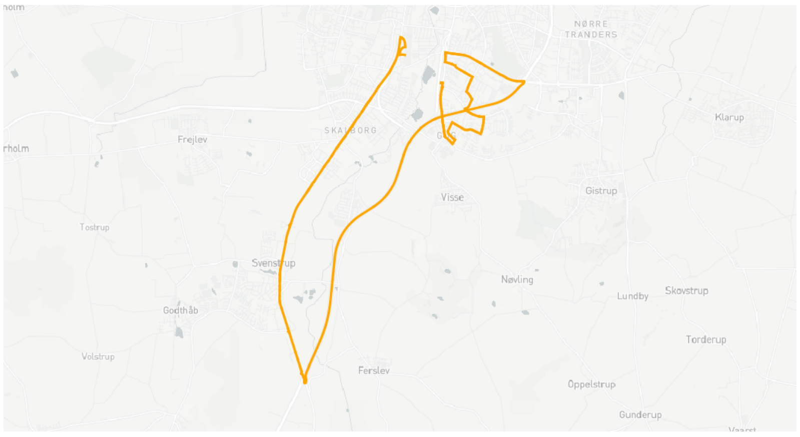


Figure 5.1: Trip Route

Figure 5.1 shows the 45-minute trip which is about 30 km. The trip started by going south on the left-hand side through *Skalborg* and *Svenstrup*, and then turning onto the E45 highway northbound.

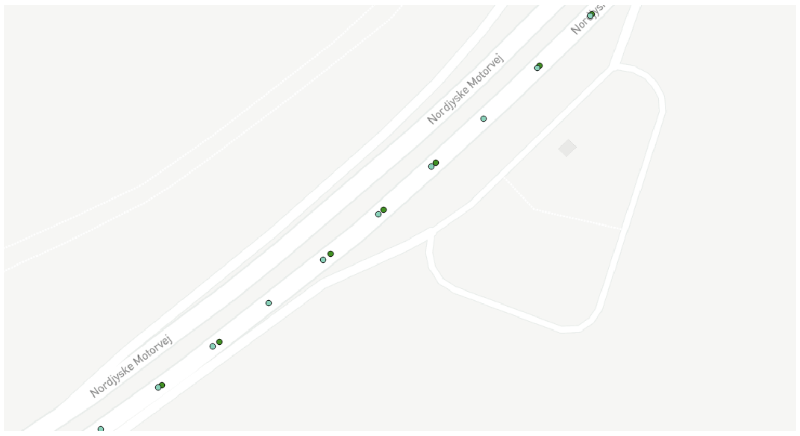


Figure 5.2: Map Matched GPS Locations (Green is Android, blue is iPhone)

Figure 5.2 shows the map-matched GPS point spread on the E45 highway from the Android and iPhone smartphones, depicted by green and blue dots, respectively.

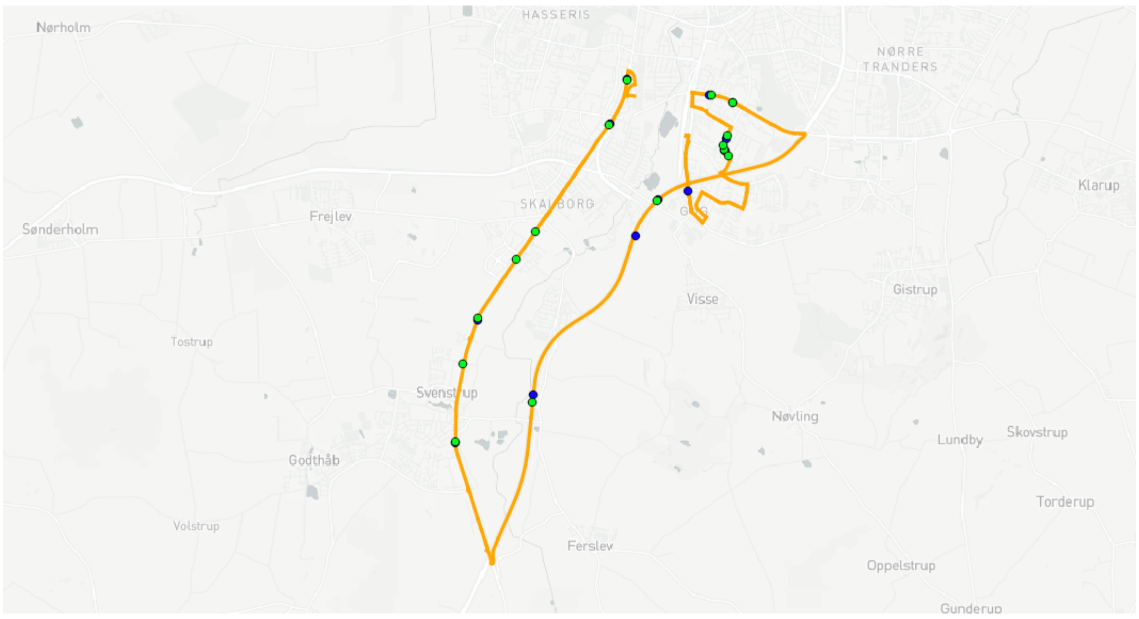


Figure 5.3: Trip Route with Notifications (Green is Android, blue is iPhone)

Figure 5.3 shows the route with dots now representing notifications from either the Android or iPhone smartphone using the same colour coding. Behind several green dots hides a blue dot, indicating both applications are alerting the user of speeding simultaneously. However, several notifications also happened individually due to wrong map matching or the speeding barely topping 10% on one application while the GPS'es are having slight differences in registered km/h.

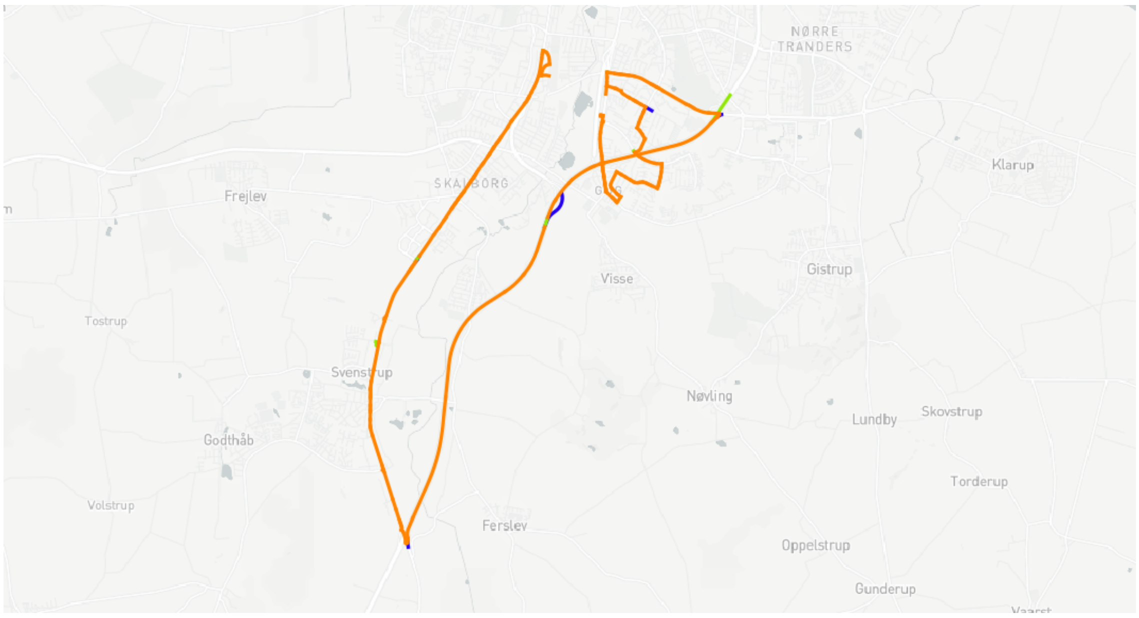


Figure 5.4: Trip Route Segments (Orange is common, green is Android, blue is iPhone)

Figure 5.4 shows the segments which GPS points was commonly and individually map matched onto. 245 road segments are in common (93.5%), Android was matched to 10 segments exclusively, and iPhone was matched to 7 exclusively. Off-line map matching would solve this issue and decrease the deviants.

Data	**Android**	**iOS**	**Difference**
Trip length	29856 metres	29995 metres	139 metres
Trip score	7.88	7.71	0.17
Average speed	39 km/t	40 km/t	1 km/t
Peak speed	139 km/t	139 km/t	0 km/t
GPS points	2638	1808	803
Sound notifications	15 x level 1 1 x level 2	10 x level 1 2 x level 2	5 x level 1 1 x level 2
Road segments	255	257	2
Battery consumption	15%	8%	7%

Table 5.1: Comparison of Trip Data

Table 5.1 presents data generated from the trip. The two platforms tracking the same trip are remarkably similar in all parameters, except for *GPS points* and *Sound notifications*. Trivial calculations reveal the Android smartphone to deliver GPS points at 0.98 Hz as opposed to the iPhone providing 0.67 Hz. The faster GPS position reading from the Android can be the reason for the sound notification differences, as the Android smartphone registered five more level 1 warnings. The faster refresh on locations could detect the speed exceeding 10% above the speed limit before dropping again while the iPhone missed this due to the lower refresh rate, as stated above. However, the calculated scores are remarkably similar, even though the two parameters GPS points and sound notifications seems significant in the score calculation. This experiment suggests that the inter-platform client application performs fairly towards the different platform users.

The battery consumption is tested under the worst-case scenario with tracking enabled, and the smartphone screen is displaying real-time updated data. The results are that Android consumes around 1% per 3rd minute and iPhone consumes 1% every 5.5 minutes which are considered acceptable.

Automatic Tracking

One issue is identified and emerges under rare conditions related to the context of Bluetooth devices mounted in the car. Cars having the accessories mode (often the first step when starting the car with a key), which many cars have as this mode is also responsible for pre-heating the engine and preparing it for ignition, turns on the Bluetooth device. If the device adheres to the conditions of automatic tracking described in Section 3.1 - *Mobile Crowd Sensing (Client Application)* it automatically connects to the smartphone. Igniting the engine from this state has been observed to restart the Bluetooth device and thereby disconnect and reconnect to the smartphone.

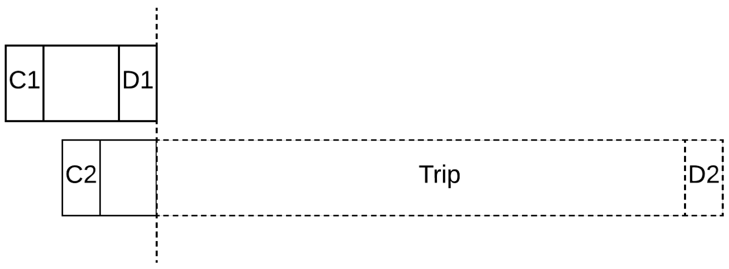


Figure 5.5: Automatic Tracking Issue

Figure 5.5 visualises the flow of the rare Bluetooth connectivity issue. Due to the occasional delay of the device disconnect, the connect intent (C2) can arrive ahead of the first disconnection (D1). In this case, C2 is disregarded as the tracking service is already running which D1 immediately after stops. The trip is not tracked under these circumstances as depicted by the dotted bar. Measures to counter this issue is delayed for further work. Otherwise, the implemented way of achieving automatic tracking is considered a viable solution for Android.

5.1.2 System Performance Testing

As mentioned in Chapter 2 - *Performance Testing*, it is important to have testing goals in mind. The testing goals in this project, are the following:

- Determine the maximum number of users able to track at the same time, while not experiencing a severe delay due to overload on the server. As the users transmit every three seconds, the response time should be within this time interval.
- Identify if there are any bottlenecks in the architecture.
- Assess whether the smartphone application's data usage is reasonable.

The approach to form the performance tests described in section Chapter 2 - *Performance Testing*, is applied in this project. In the context of tracking the users, it is mostly the process of map-matching users to the correct road as well as database interaction related to continually storing all the GPS locations. As done by Predic and Stojanovic [26], simulations of numerous vehicles were performed at the same time. GPS points from a real trip provides realistic values as input to the simulations. The length of the trip used in the simulation is 25 km. The client application sends approximately three GPS locations, using a 1 Hz GPS receiver, to the server every three seconds. The simulations replicate this to come as close to reality as possible. The performance tests were conducted on a Virtual Private Server (VPS) running Ubuntu 16.04 server with 2 GB RAM and 2 cores of an Intel(R) Xeon(R) CPU E5-2630L v2 @ 2.40 GHz. Both the server software handling the client connections and the PostgreSQL database were running on this machine. It is important to note that the performance test is conducted on the development server and not on the production server where the experiment will take place. This is done since the performance test is done alongside the user experiment and thereby could make the production server unstable, if performed in this period. The two servers are assessed to have similar specifications with the database running on a separate server in the production environment as the greatest discrepancy which may influence the performance testing results. However, they are believed to still highlight the system components that requires the most CPU power.

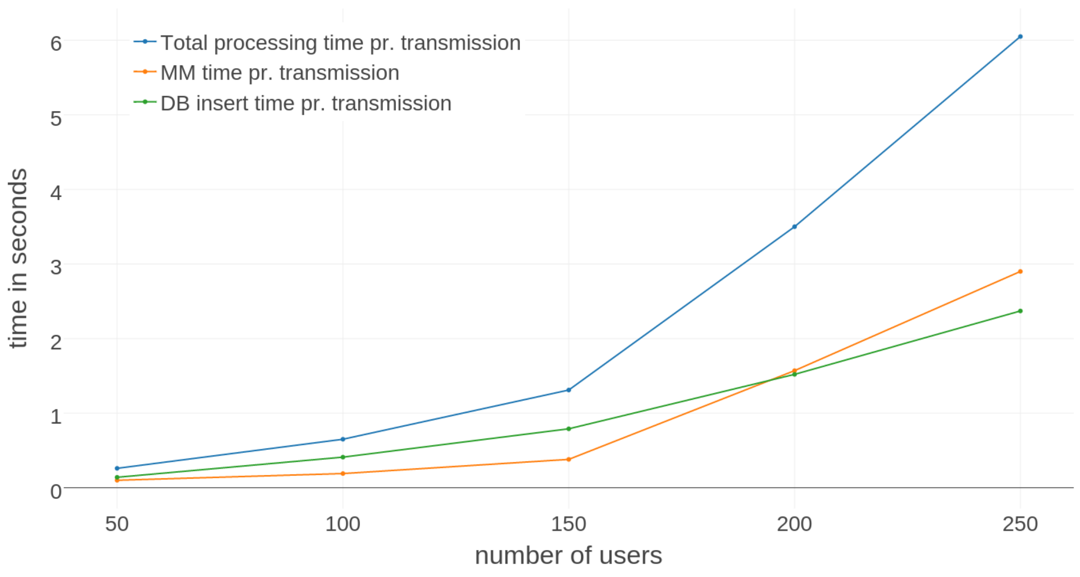


Figure 5.6: Average Server Processing Time

As seen in Figure 5.6, the total processing per data transmission from client to server is satisfiable up until 150 user, which is the maximum number able to track concurrently. Thereafter, the total processing time becomes significantly slower, with 3.5 seconds at 200 users, which will cause the users to experience a severe delay. The two most performance intensive operations on the server are map matching and database insertion time, which seem to be the causes of the slower processing time as the number of users increase. The reason why the performance worsens considerably after 150 users is most likely due to the CPU bottleneck.

The data usage for communication between the client- and server application can influence both the scalability of the system and make the application unattractive for users. Wireshark, which is a network protocol analyser [77], is applied for the data usage analysis. The analysis are based on the trip from the previous experiments. Most of the communication from client to server consists of transmitting GPS locations, where each packet has the size of 750-850 bytes. The entire trip accumulated 844 KB of data. Doing a similar trip two times every workday for a month will yield a 33.76 MB data usage, which is somewhat negligible considering the amount of data available with phone subscriptions nowadays. As described in Figure 4.10, the server sends information back to the client regarding current trip status, update of the speed limit and positive feedback regarding the user's driving. Most of the data send from the server to the client consists of trip status information where each packet has the size of 100-125 bytes. The server sends in total 180 KB of data to the client for this trip, which is also negligible.

5.2 Uncontrolled Experiments

The DriveLaB client application has been available on Apple's App Store, and Google Play Store since 10/05 – 17. The results in this section are based on the data collected from the experiment start 11/05- 17 and until 03/06 – 17. The experiment is considered uncontrolled since data contribution has been open to everyone and the project group has not had any influence on the data collected, other than encouraging people to participate. The following Table 5.2 contains an overview of the collected data along with a comparison of similar data gathered in the experiment presented in [3, p.43]:

Data	**Value**	**Last exp. value**	**Percentage**
Experiment days	23 days	32 days	-28%
Users	18 users	23 users	-22%
Trips	405 trips	435 trips	-7%
Distance	7846 km	6455 km	+22%
GPS points	534.492 points	2.764.610 points	-81%
Trips per user	22.5 trips	19 trips	+18%
Distance per user	436 km	281 km	+55%
Distance per user per day	19 km	9 km	+111%

Table 5.2: Comparison of General Data Gathered in Experiments

table 5.2 shows a decline in *Experiment days*, *Users*, *Trips* and *GPS points*, and a progress in *Distance* of 22%. The progress in distance despite a decline in all other comparable measures mainly ascribes to the increase in client application stability, as trips were being stopped prematurely in the last experiment according to user feedback. The decision to stop supporting external devices providing 10 Hz location data is accountable for the decrease of 81% in collected GPS points as only smartphones delivering 1 Hz is used in this project. Average trips and distances per user have increased and indicated more engaging users.

The distribution of trips and kilometres regarding automatic tracking are:

- **Automatic tracking:** 114 trips and 1151 km (-77%)
- **Manual tracking:** 291 trips and 6695 km (+403%)

Five users applied automatic tracking with five different Bluetooth devices registered. The percentage in parenthesis shows the kilometre difference from the experiment conducted in [3]. The proportion of trips initiated automatically to manually is not consistent with the original thesis of the project group. However, the reason can be the few Bluetooth devices, which can be due to not enough efforts to advertise this functionality in the client application. Furthermore, invitations to participate in the uncontrolled experiment happened through personal requests, which increases the chance of users willingness to please authors and thus, make an extraordinary effort to track manually.

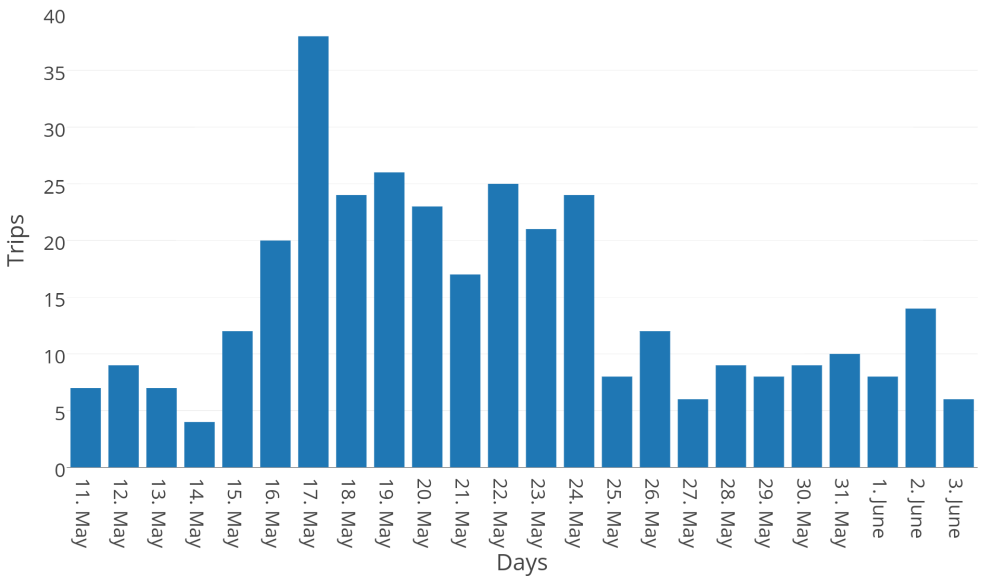


Figure 5.7: Accumulated Trips Over Experiment Days

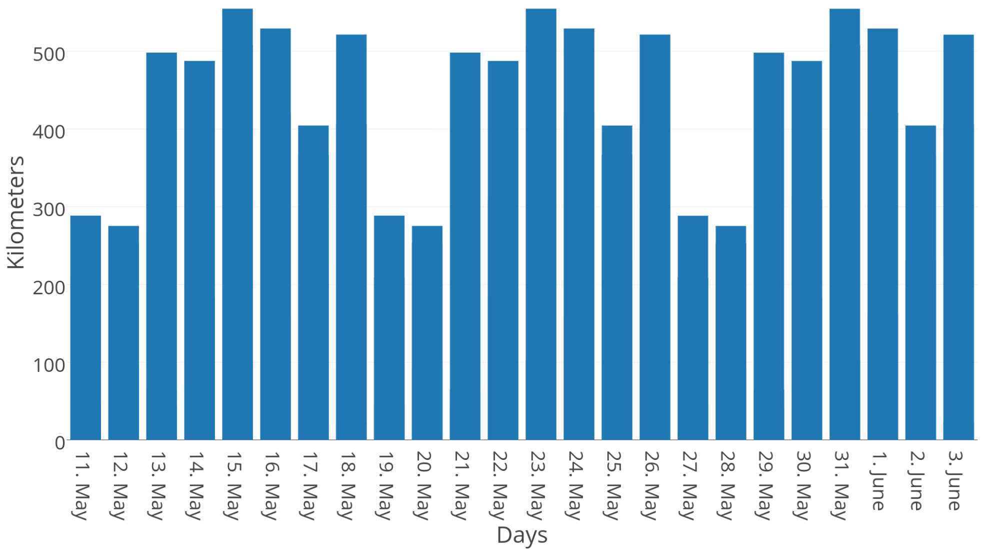


Figure 5.8: Accumulated Kilometres Over Experiment Days

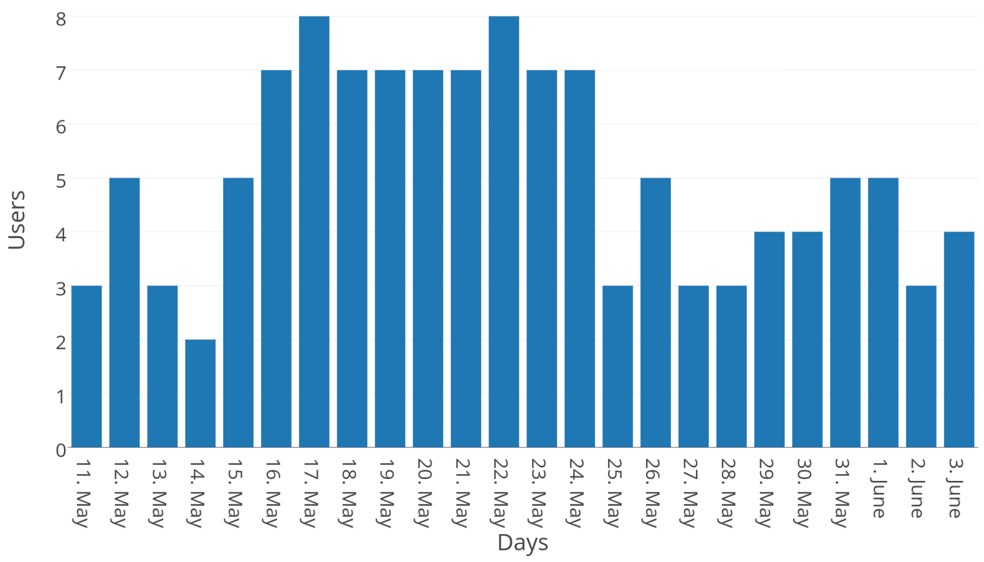


Figure 5.9: Users Tracking Over Experiment Days

Figure 5.7, 5.8 and 5.9 shows the trip, kilometre and active users over the experiment days, respectively. These figures represent the user activity throughout the experiment.

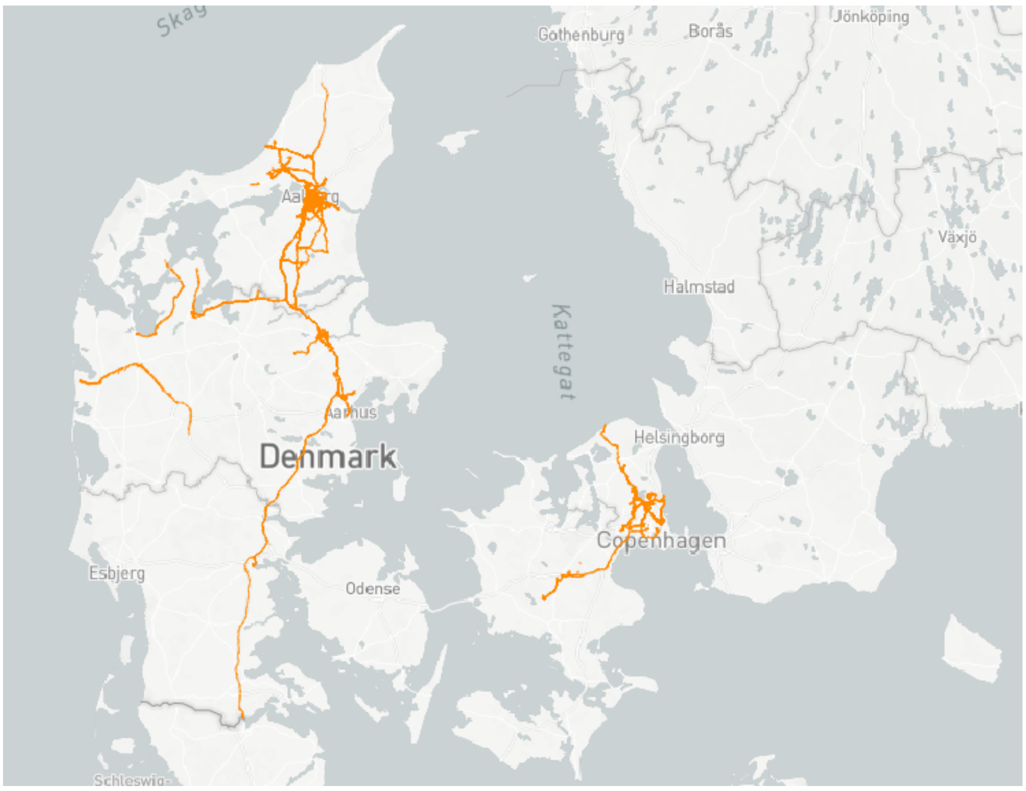


Figure 5.10: Contribution Map

Figure 5.10 illustrates the total national tracking contributions whereas four users operate in or near the Danish capital, Copenhagen. The remaining participants origins from the mainland, Jylland, and primarily in the northern part.

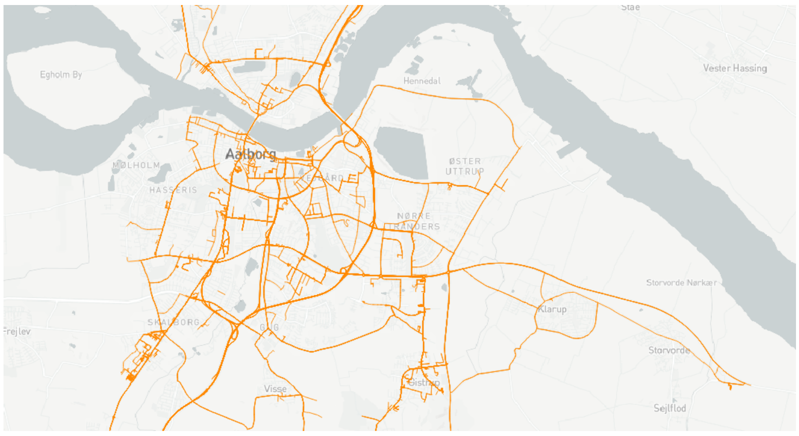


Figure 5.11: Collected Data in Aalborg

A significant part of Aalborg infrastructure has been travelled while tracking as depicted by Figure 5.11.

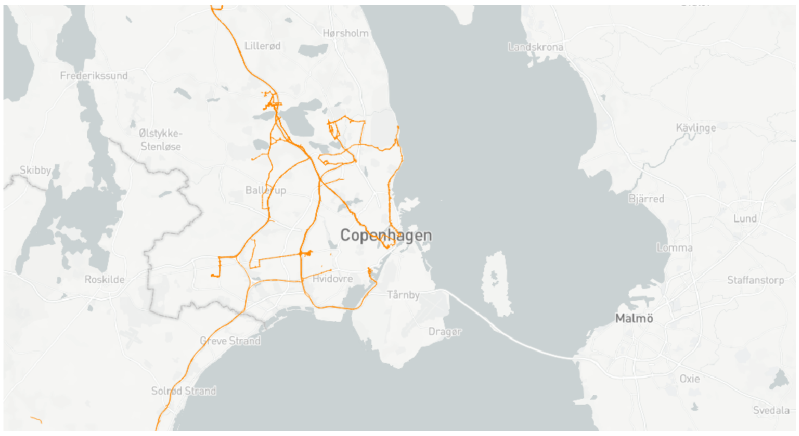


Figure 5.12: Collected Data in Copenhagen

System stability while tracking has previously been a critical issue [3] as prematurely stopping trips results in both data loss and user frustration. The DriveLaB system consist of several reliant real-time components, which increases the list of potential issues. However, the data gathered contains indications of system stability as long distance trips has been tracked flawlessly.

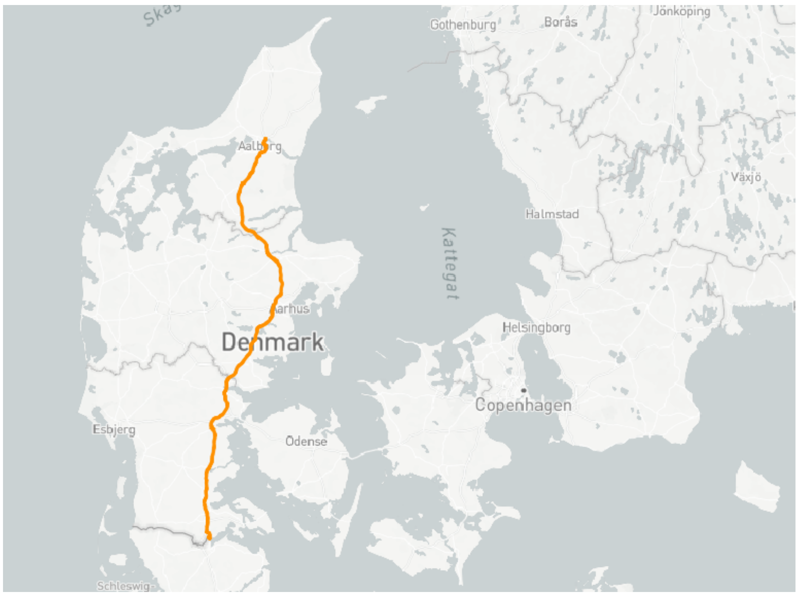


Figure 5.13: Longest trip in experiment

In Figure 5.13, the longest trip recorded is shown, ranging 301 km and lasted 2 hours and 40 minutes, which is a testimony to the system stability.

5.2.1 Speed Limits

Results of the contributed speed limit reports from the uncontrolled experiment are presented in this section.

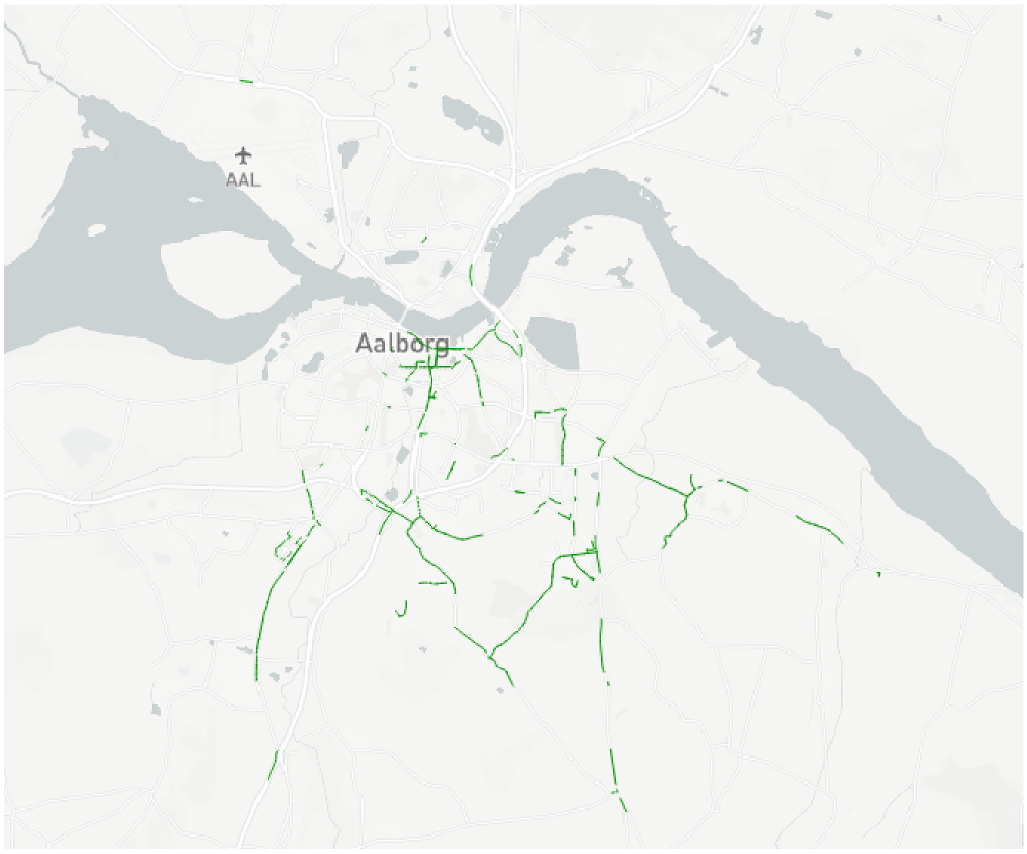


Figure 5.14: Speed Limit Reports and Their Impact

Figure 5.14 shows a map where speed limits have been altered due to speed limit reports conducted by users. There have been 11 different users providing speed limit reports, however, two users have contributed with the vast majority. Almost all changes have occurred within the vicinity of Aalborg.

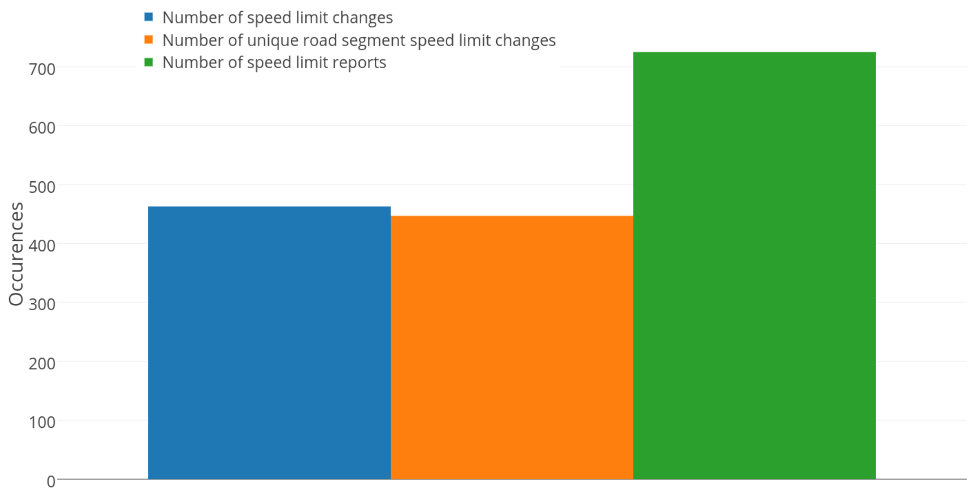


Figure 5.15: Distribution of Speed Limit Reports and Changes

The distribution of speed limit changes made in total and to unique road segments in addition to the number of total speed limit reports is shown in Figure 5.15. Most changes made to a road have been permanent throughout the test. This is expected as the experiment is only two weeks and there are relatively few participants. Additionally, the participants in the experiment do most likely not have malicious intents so there is no incentive to report incorrect speed limits that would need to be reverted. The ratio between speed limit changes made to a road segment and number of speed limit reports contributed is $\frac{1}{1.56}$. This might be explained by the reputation system being fairly lax on new users in addition to the low number of participants contributing with most of the speed limit reports, thereby gaining a good reputation score.

		New speed limit										
		20	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**	**110**	**130**
Previous speed limit	**20**	0	0	0	0	0	0	0	0	0	0	0
	30	0	0	0	4	0	0	0	0	1	0	0
	40	0	0	0	1	0	0	1	0	0	0	0
	50	2	17	20	0	2	0	3	0	0	0	0
	60	0	0	2	6	0	1	4	0	0	0	0
	70	0	0	0	3	2	0	2	0	0	0	0
	80	0	1	32	247	89	10	0	0	0	2	0
	90	0	0	0	0	0	0	0	0	0	1	0
	100	0	0	0	0	0	0	0	0	0	0	0
	110	0	0	0	0	0	0	0	1	0	0	0
	130	0	0	0	0	0	0	0	0	0	9	0

Table 5.3: Speed Limit Change Matrix

The matrix in Table 5.3 shows what speed limits have been changed from and to. The blue boxes with a number indicate the number of occurrences, where the speed limit have changed from a higher speed to a lower speed, thereby requiring the users to drive slower to get a good performance score. The red boxes indicate the number occurrences where speed limits have been changed from a lower to a higher speed limit. Surprisingly an overwhelming majority of changes have been from a higher to a lower speed limit, in particular from 80 km/h down to 50 and 60 km/h. This is due to the OSM road classifications, as most of those roads are tagged with road class 31 that translates to tertiary roads, which have a default speed limit of 80 km/h. This shows that some of the road classifications cover too many different speed limits, thus it becomes unreliable to use and another way of classifying should be considered. There have been very few speed limits on the motorways, which could indicate that the contributors either have driven less on that road type or the speed limits for those road segments are already well mapped in OSM. The speed limit change from 30 to 100 km/h is most likely an error, as the road is looked up through *Google Street View* where there is no traffic sign indicating a speed limit of 100 km/h. Other conspicuous road changes are those changing from 80 to 110 and 90 to 110, these are however all correct.

5.2.2 Notifications

Speeding notifications are implemented to improve the drivers' awareness and thereby, hopefully improve road safety. Whether the driver reacts to these notifications in a positive manner, is what will be investigated in this section. In [3], phone volume was not tracked, rendering it impossible to guarantee if notifications got played through the speaker or not. However, phone volume is only one out of many variables needed to be considered, to give credible results. Therefore, the following list of criteria is followed, to accommodate unbiased results

- Sound level must be 5 or higher.
- Notifications near junctions are discarded as de-acceleration is already expected.
- Notifications triggered by incorrect speed limits are disregarded.

Contextual information such as congestion levels and road obstacles are also relevant variables to include, however, such data is not available for this project and therefore not considered.

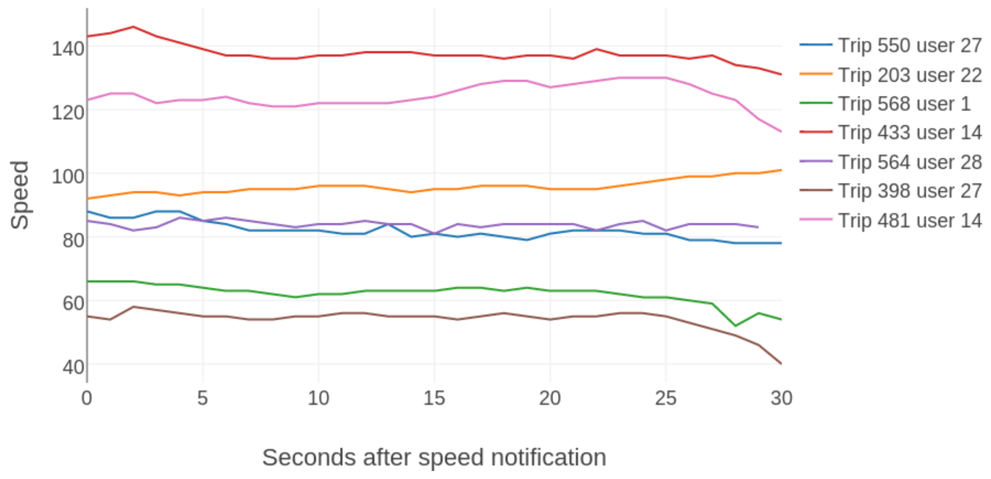


Figure 5.16: Speed Trend After Notifications

Figure 5.16 shows seven selected trips, meeting the aforementioned requirements. Trip 433, 481, 550, 568, and 398 all de-accelerates within 30 seconds where trip 564 remains unchanged and 203 increases in velocity. All trips with decreasing velocity end up near the displayed speed limit, which could indicate that contextual implications are absent. Based on the data presented in Figure 5.16, notifications seems to decrease the speed in the 30 seconds period. A greater sample size should be evaluated to establish a stronger conclusion.

5.2.3 Driver Score Evolvement

It is decided to look at the involvement of trip scores to find if drivers improve over time. Each score is calculated, such that the 11th of May is the first possible data entry used for the score calculation. Furthermore, only a history of three days prior the measured element is used to form the score. This way, the score is less sensitive to sudden changes and thereby better outlines trending patterns. The three-day evaluation resembles the leaderboard logic of active users, which achieves a more general perspective on the question of user driving improvement. However, increasing the day count is not ideal because of the short experiment period.

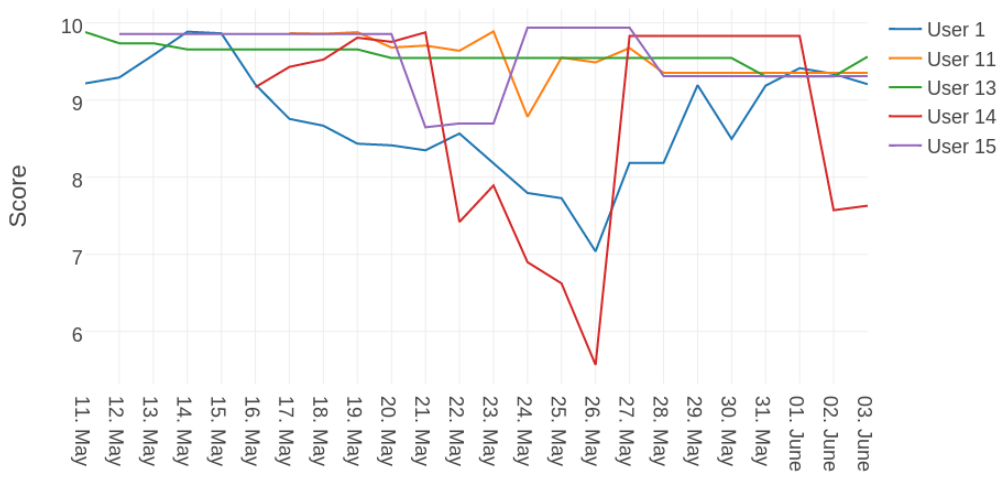


Figure 5.17: Score Evolvement

Unfortunately, no significant sign of improvement can be deduced from Figure 5.17. Monitoring a longer period may return different results as 23 days may not be enough to affect driver habits acquired over a long period. Additionally, more convincing rewards associated with the score could lead to a greater effort regarding improving the score. As Figure 5.17 use three historical days to base the score on, the individual days cannot be used to pinpoint events.

5.2.4 Men vs. Women

Gender is specified by the user upon signing up to an account in the client application. This knowledge can be used to investigate driving performance between the genders. There has been 10 men and 8 women signed up for DriveLaB during the experiment period.

User Id	**Gender**	**Total Score**	**Kilometres tracked**
25	Male	9.94	130 km
13	Male	9.68	216 km
11	Male	9.61	790 km
32	Female	9.29	54 km
18	Female	9.24	937 km
34	Female	9.15	16 km
15	Female	9.10	309 km
19	Female	8.95	164 km
14	Male	8.62	2.318 km
1	Male	8.56	437 km
33	Male	8.54	747 km
28	Male	8.45	420 km
27	Female	7.97	353 km
31	Male	7.89	220 km
30	Female	6.34	735 km
26	Female	4.32	466 km
22	Male	3.85	57 km
21	Male	2.28	88 km

Table 5.4: Contributers specified with gender, score and kilometres tracked

A gender total score can be calculated by adopting the same logic as the one to calculate each person's total score, which means that a person's score is affecting the total gender score relational to kilometres driven. This results in the following total scores:

1. Male: 8.65
2. Women: 7.59

5.3 User Feedback

This section relies solely on informal statements received by test participants. Therefore, these statements are used as an indication of the system impression in general.

Incorrect Speed Limits

User 11, 25, 26, 27, 28, and 34 all mentions that the application often displays incorrect speed limits. One user experienced getting an undeserved penalty point notification, where another states how the incorrect speed limits decrease the trustworthiness of the application. Additionally, user 11 remarks that the application displays a wrong street name and speed limit at some locations.

Obscure Speed Limit Reporting System

User 26, 27, and 34 are confused by speed limit reports not causing changes immediately. They experience this as an error in the system.

User 11 points out that the reporting instruction overlay blocks functionalities, which causes frustration. Furthermore, user 25 argues that the reporting system requires too much attention during a drive. This is due to the combination of touching the smartphone screen and commanding speed limit changes through voice. The speech recognition has difficulties capturing the commands in some circumstances, noticed by both user 11 and 25.

Incorrect Driving Distance

User 25 highlights how identical routes gets different distance readings. This causes confusion as it is unexpected behaviour.

Automatic Tracking

User 14 deactivates Bluetooth and GPS while not using the *Client Application* to save battery power on work. The user elaborates that the car is already running before remembering to turn back on the Bluetooth and GPS, resulting in automatic tracking not being utilised. Both user 14 and 27 concludes that automatic tracking is reliable when used as prescribed.

Reflections 6

Discussion

An initial objective of the project was to identify a solution to enable automatic tracking. The implemented solution offers a quite reliable approach to automatic tracking. However, as pointed out by user 14 in Section 5.3 - *User Feedback*, having Bluetooth and GPS enabled at all times is unappealing due to the potential battery drainage. This needs to be investigated further, as it questions the viability of the solution. However, it can be argued that the target group of frequent drivers for the client application are inclined to have their smartphones docked and powered while driving, rendering the issue of minimal impact.

Speed limits play a central role in the system and currently dominates the user feedback in Section 5.3 - *User Feedback*. Seven users mention the incorrect display of speed limits where several potential causes to this issue are possible:

- *Map matching* - The user is mapped to an incorrect road segment. The map matching could be more cautious of announcing a road segment change.
- *Incorrect road splitting* - Road segments are divided inappropriately in relation to speed limits. Road segments could be divided into smaller portions to relieve this problem.
- *Slow response time* - The user only receives new speed limit updates every 3 seconds which can cause some delay when a road segment with the new speed limit is encountered. This could be improved by storing the map client-side, thereby providing much faster response time and better scalability as described in Section 3.3.2 - *Map Matching*.
- *Wrong road classification* - Unknown speed limits are evaluated by a heuristic approach, which often delivers an incorrect speed limit, as indicated by the results in Section 5.2.1 - *Speed Limits*. A better approach might be to use a map that divides Denmark into city, rural, and holiday neighbourhoods to deduce speed limits.

Transparency in the speed limit report processing could also provide a positive experience for users and increase the understanding of how speed limits are verified and not instantaneously accepted. Also, the reporting includes minimal, but still illegal, smartphone interaction which questions the moral viability of this functionality.

Conclusion

This study set out to investigate and implement a speeding reductive MCS platform. Throughout this report, solutions to several prespecified platform requirements have been designed, with their implementation documented. Exposing the driver to auditory speeding notifications shows promising results, and could be worthwhile exploring further.

The presented solution to enable automatic tracking on Android using Bluetooth devices commonly found in vehicles performs well, however, is threatened by user initiatives to conserve battery consumption. Unfortunately, it is unknown whether iPhone supports an implementation of automatic tracking, utilising generic Bluetooth devices. The approach of allowing users to report speed limit corrections seems to function as intended. However, feedback indicates that evaluating the reports must take place more often.

The DriveLaB smartphone application did perform equally well on both the Android and iPhone platform supporting the viability of developing a cross-platform application using the Xamarin platform. Also, encryption is implemented to obfuscate the personal data interchanged. This offers a more secure overall system as the transferred data is now unusable for malicious users.

The system has been subject to both controlled and uncontrolled experiments and has through these been used to track over 7800 km of driving distance. The results of these experiments indicate:

- Inter-platform fairness.
- Acceptable client application battery consumption.
- Possible system performance bottlenecks.
- A limit of 150 simultaneously tracking users with adequate system responsiveness.
- That OSM road classifications are insufficient in regard to deduce speed limits.
- Males drive slightly less above the speed limit than women.
- Speeding notifications decrease speeding.

The system also enables CSR opportunities through a public API, whereas an application consuming the API is developed as a demonstration. It shows speed related information and proves the usefulness of the public API.

Future work

This section presents the suggestions of future work with the perspective of turning DriveLaB into a commercial platform. In this setting, there are three categories of future work identified; *Scalability*, *Viability* and *Data Analysis*.

Scalability is an important issue for commercialisation of the platform and focuses should primarily be directed at increasing the amount of simultaneously tracking users. Relocating the road map and calculations of map matching and driving information from the server to the client application is considered the most influential scalability factor. This introduces the complex task of keeping client application databases synchronised always to have an updated roadmap.

Viability includes investigating stronger user incentives besides offering driving related information which is considered only to attract the enthusiast users. The development of key partner protected APIs to enable CSR initiatives on behalf of socially aware companies could provide sustainability to the system. Lastly, a solution to provide automatic tracking on iOS is necessary to sustain regular user participation.

Data analysis has the potential to deduce valuable information from the data collected given the complexity of the driving domain. Further research should be undertaken to investigate the possibility of extracting speed limits using machine intelligence to avoid the illegal user smartphone interaction while driving. Another path is maintaining the report system, which would then include testing, and researching improvements, for the reputation system.

Acknowledgement

Thanks to all the 18 drivers that participated in the 23-day long experiment, rendering the system evaluation possible. Also thanks to *Center for Data-Intensive Cyber-Physical Systems* `www.dicyps.dk` for applying the necessary prerequisites for the project. *AAU IT Services* (ITS) must also be thanked for supplying a MacBook rendering iOS development possible. Lastly, the project group is grateful for the valuable guidance offered throughout the project by Kristian Torp.

Bibliography

[1] Casper Holst Laustsen Morten Møller Jacobsen and Johan Leth Gregersen. Aalborg university report: An advanced usage based insurance and privacy-secure pricing model. 2015-2016.

[2] Casper Holst Laustsen Morten Møller Jacobsen and Johan Leth Gregersen. Aalborg university report: Drive-lab: An experimental platform for usage-based car insurance. 2016.

[3] Kasper Fromm Pedersen Dennis Rasmussen and Thomas Frisk Olsen. Aalborg university report: Drivelab - a driver behavioral information system. 2016-2017.

[4] Glo$^{\text{TM}}$ | garmin. `https://buy.garmin.com/da-DK/DK/p/109827`. (Accessed on 08/06/2017).

[5] Double battery beacon. `https://store.kontakt.io/our-products/30-double-battery-beacon.html`. (Accessed on 08/06/2017).

[6] Vejdirektoratet Årsstatistik. URL `http://vejdirektoratet.dk/DA/viden_og_data/statistik/ulykkestal/%c3%85rsstatistik/Sider/default.aspx`. (Accessed on 08/06/2017).

[7] Dr article - price of a road traffic accident. URL `https://www.dr.dk/nyheder/indland/pris-et-trafikuheld-600000-kroner`. (Accessed on 08/06/2017).

[8] Safe traffic - accidents cost the society big time. URL `https://www.sikkertrafik.dk/media/2868/ulykker-koster-samfundet-kassen.pdf`. (Accessed on 08/06/2017).

[9] Road safety commission - plan of action. URL `http://www.faerdselssikkerhedskommissionen.dk/sites/kombelt.dev2.1508test.dk/files/filer/Handlingsplan%202013-2020%20Hver%20ulykke%20er%20%C3%A9n%20for%20meget%20-%20et%20f%C3%A6lles%20ansvar.pdf`. (Accessed on 08/06/2017).

[10] Tryg fonden - sikker i trafikken, . URL `https://www.trygfonden.dk/fokus/sikkerhed/sikker-i-trafikken`. (Accessed on 08/06/2017).

[11] Tryg fonden - donations, . URL `https://www.trygfonden.dk/searchresultpage#!npt=donation&currentPage=1&Sort=-year&FocusArea=SikkerITrafikken`. (Accessed on 08/06/2017).

[12] Rådet for sikker trafik. URL `https://www.sikkertrafik.dk/samarbejde/trygfonden`. (Accessed on 08/06/2017).

[13] København open data trafikhastigheder. URL `http://data.kk.dk/dataset/trafikhastigheder`. (Accessed on 08/06/2017).

[14] Google roads api terms, . URL `https://developers.google.com/maps/terms`. (Accessed on 08/06/2017).

[15] Open street map. URL `https://www.openstreetmap.org`. (Accessed on 08/06/2017).

[16] Bin Guo, Zhiwen Yu, Xingshe Zhou, and Daqing Zhang. From participatory sensing to mobile crowd sensing. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pages 593–598. IEEE, 2014.

[17] Raghu K Ganti, Fan Ye, and Hui Lei. Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine*, 49(11), 2011.

[18] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. Cartel: a distributed mobile sensor computing system. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 125–138. ACM, 2006.

[19] Prashanth Mohan, Venkata N Padmanabhan, and Ramachandran Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 323–336. ACM, 2008.

[20] B Thomas Adler and Luca De Alfaro. A content-driven reputation system for the wikipedia. In *Proceedings of the 16th international conference on World Wide Web*, pages 261–270. ACM, 2007.

[21] Afra J Mashhadi and Licia Capra. Quality control for real-time ubiquitous crowdsourcing. In *Proceedings of the 2nd international workshop on Ubiquitous crowdsouring*, pages 5–8. ACM, 2011.

[22] Maryam Pouryazdan, Burak Kantarci, Tolga Soyata, Luca Foschini, and Houbing Song. Quantifying user reputation scores, data trustworthiness, and user incentives in mobile crowd-sensing. *IEEE Access*, 5:1382–1397, 2017.

[23] Mahdi Hashemi and Hassan A Karimi. A critical review of real-time map-matching algorithms: Current issues and future directions. *Computers, Environment and Urban Systems*, 48:153–165, 2014.

[24] Nagendra R Velaga, Mohammed A Quddus, and Abigail L Bristow. Developing an enhanced weight-based topological map-matching algorithm for intelligent transport systems. *Transportation Research Part C: Emerging Technologies*, 17(6):672–683, 2009.

[25] Elaine J Weyuker and Filippos I Vokolos. Experience with performance testing of software systems: issues, an approach, and case study. *IEEE transactions on software engineering*, 26(12):1147–1156, 2000.

[26] Bratislav Predic and Dragan Stojanovic. Enhancing driver situational awareness through crowd intelligence. *Expert Systems with Applications*, 42(11):4892–4909, 2015.

[27] Vejdirektoratet Årsstatistik, . URL `http://nyheder.tv2.dk/business/2017-04-08-tryg-indforer-overvagning-af-sine-bliforsikringskunder`. (Accessed on 08/06/2017).

[28] Philippe Baecke and Lorenzo Bocca. The value of vehicle telematics data in insurance risk selection processes. *Decision Support Systems*, pages –, 2017. ISSN 0167-9236. doi: https://doi.org/10.1016/j.dss.2017.04.009. URL `http://www.sciencedirect.com/science/article/pii/S0167923617300763`.

[29] Ionic 2. URL `http://ionicframework.com/docs/intro/installation/`. (Accessed on 08/06/2017).

[30] Xamarin platform. URL `https://www.xamarin.com/platform`. (Accessed on 08/06/2017).

[31] Android Documentation. Broadcast receivers, . URL `https://developer.android.com/reference/android/content/BroadcastReceiver.html`. (Accessed on 08/06/2017).

[32] Android Documentation. Bluetooth devices, . URL `https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html`. (Accessed on 08/06/2017).

[33] Android Documentation. Services, . URL `https://developer.android.com/guide/components/services.html`. (Accessed on 08/06/2017).

[34] Microsoft. Signalr, . URL `http://signalr.net/`. (Accessed on 08/06/2017).

[35] Jon Postel. Transmission control protocol. 1981.

[36] Jon Postel. User datagram protocol. Technical report, 1980.

[37] I. Fette and A. Melnikov. The websocket protocol. RFC 6455, RFC Editor, December 2011. URL `http://www.rfc-editor.org/rfc/rfc6455.txt`. `http://www.rfc-editor.org/rfc/rfc6455.txt`.

[38] Microsoft. Msdn - data transfer objects, . URL `https://msdn.microsoft.com/en-us/library/ms978717.aspx`. (Accessed on 08/06/2017).

[39] Fartgrænser i bil. URL `http://www.fdm.dk/biler/love-regler/fartgraenser-bil`.

[40] Siniša Husnjak, Dragan Peraković, Ivan Forenbacher, and Marijan Mumdziev. Telematics system in usage based motor insurance. *Procedia Engineering*, 100: 816–825, 2015.

[41] Burak Kantarci and Hussein T Mouftah. Trustworthy sensing for public safety in cloud-centric internet of things. *IEEE Internet of Things Journal*, 1(4):360–368, 2014.

[42] Restful api versioning insights. `http://blog.restcase.com/restful-api-versioning-insights/`. (Accessed on 08/06/2017).

[43] Api versioning methods, a brief reference. `https://www.3scale.net/2016/06/api-versioning-methods-a-brief-reference/`, . (Accessed on 08/06/2017).

[44] Api versioning methods, a brief reference - dzone integration. `https://dzone.com/articles/api-versioning-methods-a-brief-reference`, . (Accessed on 08/06/2017).

[45] Apigility. `https://apigility.org/documentation/api-primer/versioning`. (Accessed on 08/06/2017).

[46] Bing maps. `https://msdn.microsoft.com/en-us/library/dd877180.aspx`. (Accessed on 08/06/2017).

[47] Google maps apis | google maps apis | google developers. `https://developers.google.com/maps/documentation/`, . (Accessed on 08/06/2017).

[48] Geojson. `http://geojson.org/`, . (Accessed on 08/06/2017).

[49] Documentation - leaflet - a javascript library for interactive maps. `http://leafletjs.com/reference-1.0.3.html#geojson`. (Accessed on 08/06/2017).

[50] Data layer | google maps javascript api | google developers. `https://developers.google.com/maps/documentation/javascript/datalayer#load_geojson`. (Accessed on 08/06/2017).

[51] Geojson module examples. `https://msdn.microsoft.com/en-us/library/mt750522.aspx`, . (Accessed on 08/06/2017).

[52] Xamarin. Xamarin forms - build ui with xaml, . URL `https://www.xamarin.com/forms`. (Accessed on 08/06/2017).

[53] Xamarin. Documentation on shared projects, . URL `https://developer.xamarin.com/guides/cross-platform/application_fundamentals/shared_projects/`. (Accessed on 08/06/2017).

[54] GitHub. Xamarin github official plugins, . URL `https://github.com/xamarin/XamarinComponents`. (Accessed on 08/06/2017).

[55] GitHub. James montemagno official xamarin geolocator plugin, . URL `https://github.com/jamesmontemagno/GeolocatorPlugin`. (Accessed on 08/06/2017).

[56] GitHub. Geolocator plugin issue #54, . URL `https://github.com/jamesmontemagno/GeolocatorPlugin/issues/54`. (Accessed on 08/06/2017).

[57] GitHub. Geolocator plugin issue #76, . URL `https://github.com/jamesmontemagno/GeolocatorPlugin/issues/76`. (Accessed on 08/06/2017).

[58] GitHub. Geolocator plugin issue #75, . URL `https://github.com/jamesmontemagno/GeolocatorPlugin/issues/75`. (Accessed on 08/06/2017).

[59] Microsoft. Singleton pattern, . URL `https://msdn.microsoft.com/en-us/library/ff650316.aspx`. (Accessed on 08/06/2017).

[60] Android. Android manifest documentation, 2016. URL `https://developer.android.com/guide/topics/manifest/manifest-intro.html`.

[61] Rick-Anderson. Introduction to asp.net core | microsoft docs, 2016. URL `https://docs.microsoft.com/en-us/aspnet/core/`.

[62] rowanmiller. Entity framework core | microsoft docs, 2016. URL `https://docs.microsoft.com/en-us/ef/core/`.

[63] C Carl Robusto. The cosine-haversine formula. *The American Mathematical Monthly*, 64(1):38–40, 1957.

[64] www.movable-type.co.uk Chris Veness. Calculate distance and bearing between two latitude/longitude points using haversine formula in javascript. URL `http://www.movable-type.co.uk/scripts/latlong.html`.

[65] Current weather and forecast - openweathermap. `https://openweathermap.org/`. (Accessed on 08/06/2017).

[66] Bil | trafiktypen.dk. `http://www.trafiktypen.dk/bil`. (Accessed on 08/06/2017).

[67] Geofabrik // home. `http://www.geofabrik.de/`. (Accessed on 08/06/2017).

[68] osm2po - openstreetmap converter and routing engine for java. `http://osm2po.de/`. (Accessed on 08/06/2017).

[69] 2017. URL `https://www.python.org/`.

[70] Postgresql: Documentation: 9.4: Pl/python - python procedural language.

[71] pgagent — pgadmin iii 1.22.2 documentation.

[72] Unoeuro - webhoteller og domæner. `https://www.unoeuro.com/`. (Accessed on 08/06/2017).

[73] Leaflet - a javascript library for interactive maps. `http://leafletjs.com/`. (Accessed on 08/06/2017).

[74] Documentation - materialize. `http://materializecss.com/`. (Accessed on 08/06/2017).

[75] jquery. `https://jquery.com/`. (Accessed on 08/06/2017).

[76] Mark D Fairchild. *Color appearance models.* John Wiley & Sons, 2013.

[77] Wireshark · go deep, 2017. URL `https://www.wireshark.org/`.

Data Warehouse Schema A

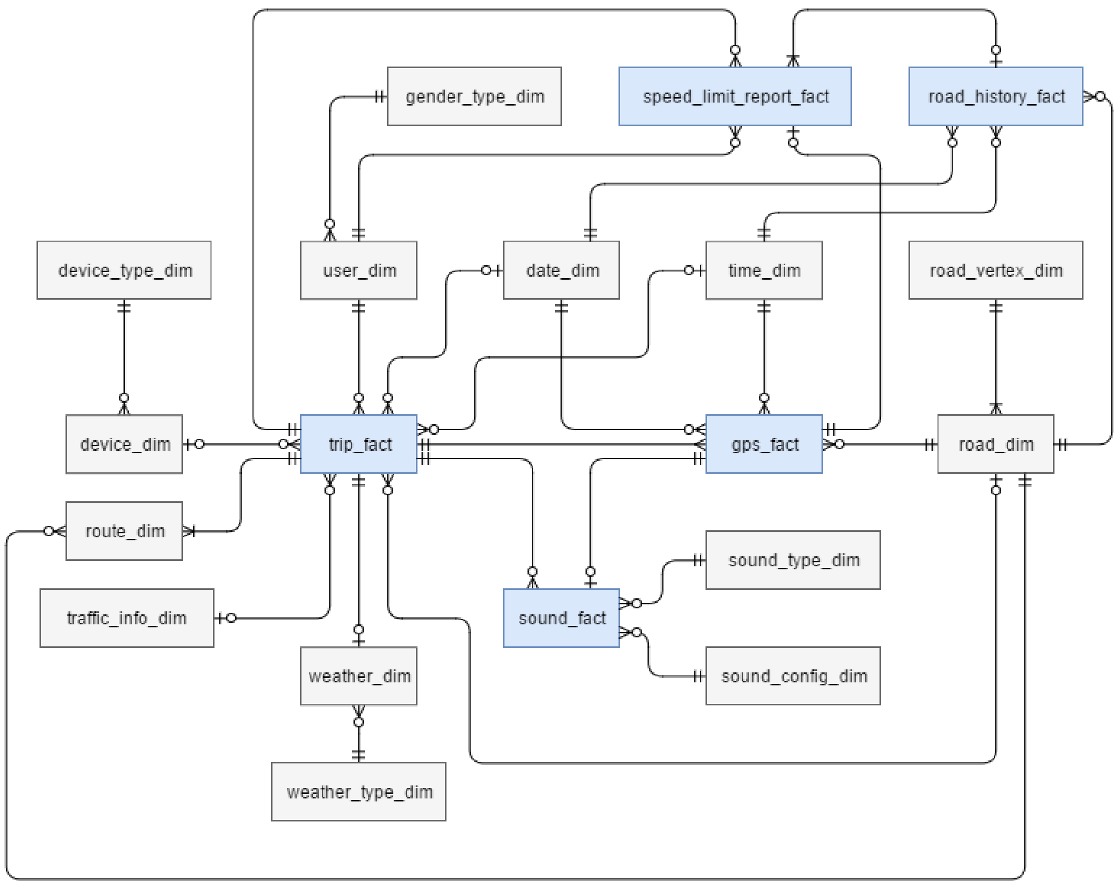


Figure A.1: Entity relationship diagram using Crow's Foot notation representing the DriveLaB database structure

A.1 Fact Tables

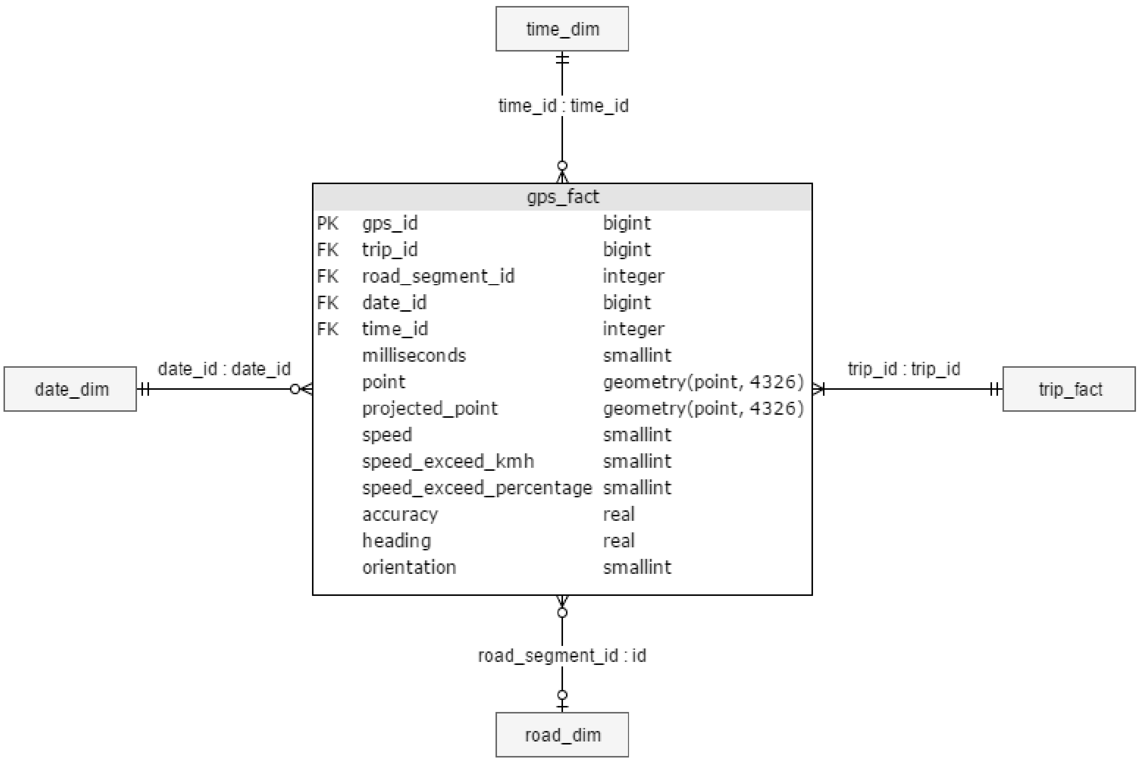


Figure A.2: GPS Fact Table

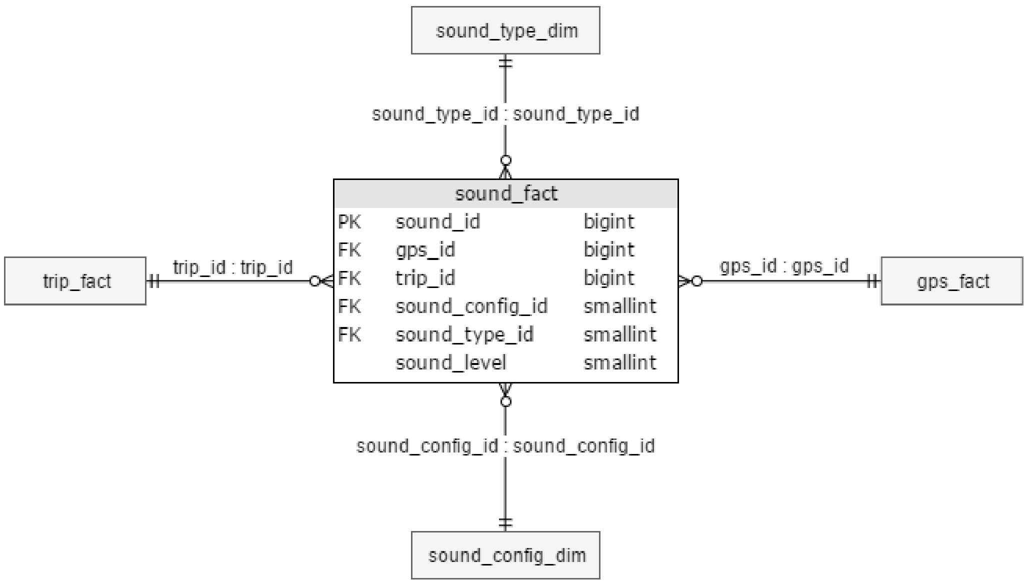


Figure A.3: Sound Fact Table

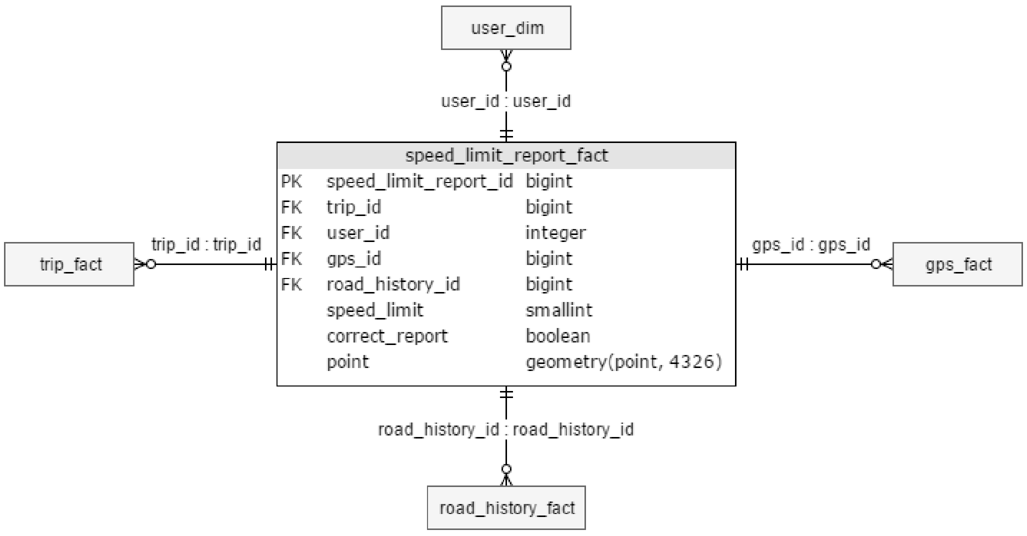


Figure A.4: Speed Limit Report Fact Table

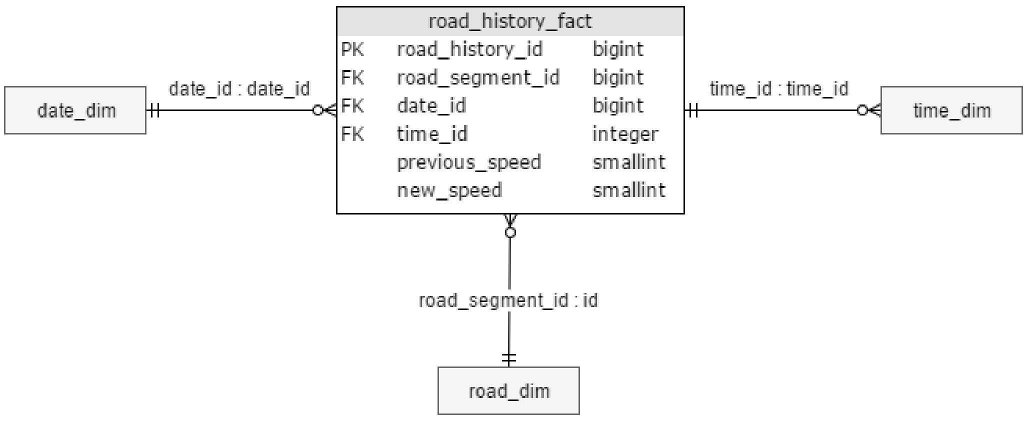


Figure A.5: Road History Fact Table

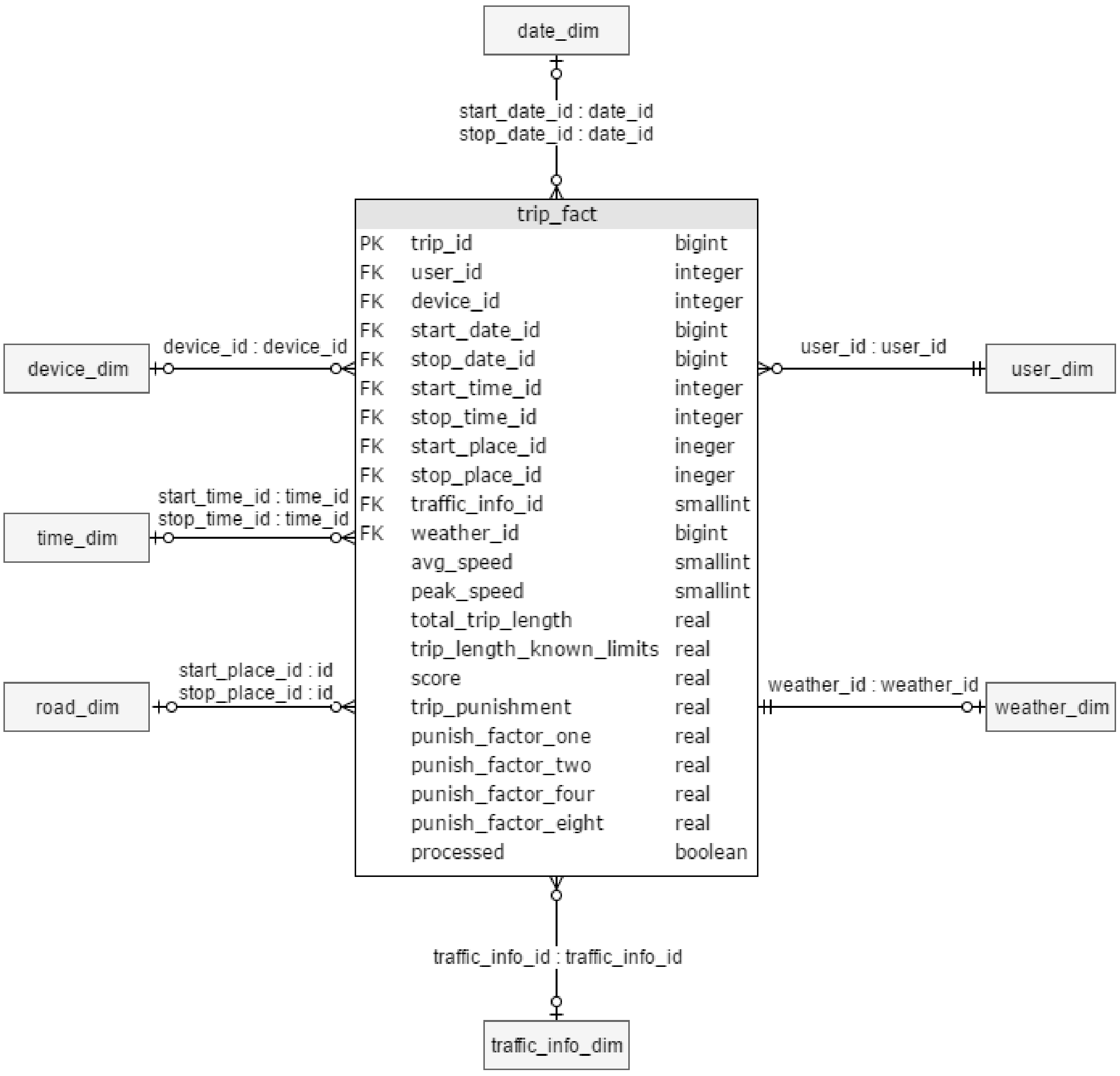
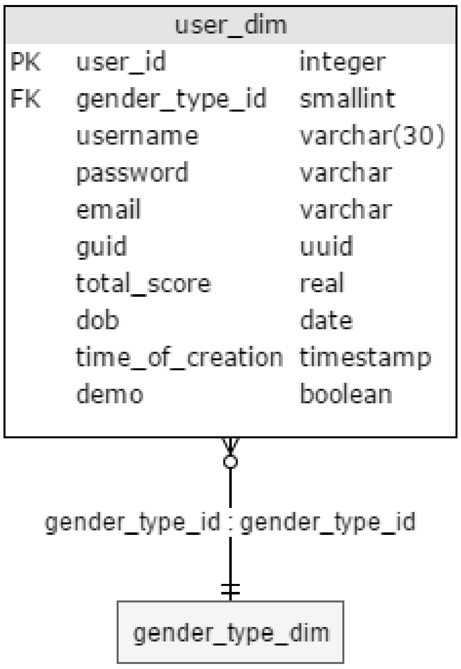


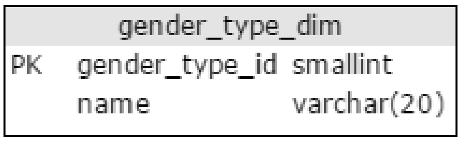
Figure A.6: Trip Fact Table

A.2 Dimension Tables



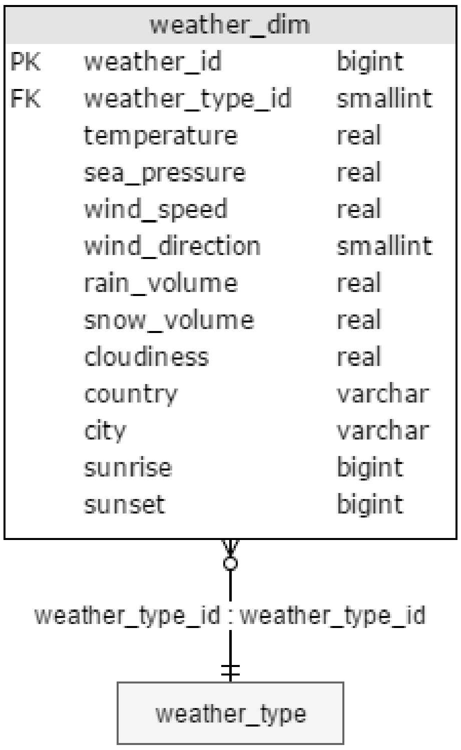
user_dim		
PK	user_id	integer
FK	gender_type_id	smallint
	username	varchar(30)
	password	varchar
	email	varchar
	guid	uuid
	total_score	real
	dob	date
	time_of_creation	timestamp
	demo	boolean

Figure A.7: User Dimension Table



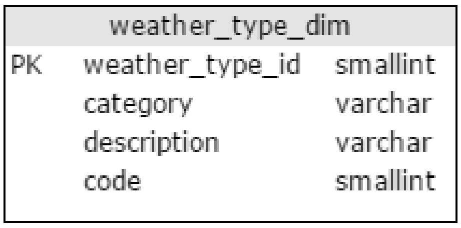
gender_type_dim		
PK	gender_type_id	smallint
	name	varchar(20)

Figure A.8: Gender Type Dimension Table



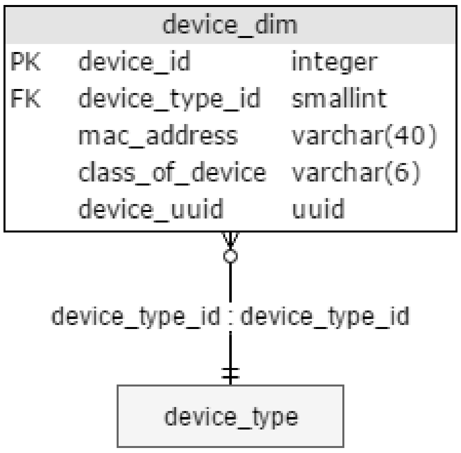
weather_dim		
PK	weather_id	bigint
FK	weather_type_id	smallint
	temperature	real
	sea_pressure	real
	wind_speed	real
	wind_direction	smallint
	rain_volume	real
	snow_volume	real
	cloudiness	real
	country	varchar
	city	varchar
	sunrise	bigint
	sunset	bigint

Figure A.9: Weather Dimension Table



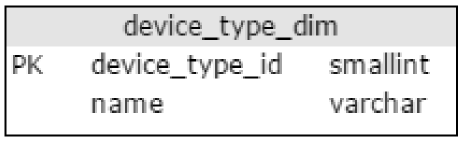
weather_type_dim		
PK	weather_type_id	smallint
	category	varchar
	description	varchar
	code	smallint

Figure A.10: Weather Type Dimension Table



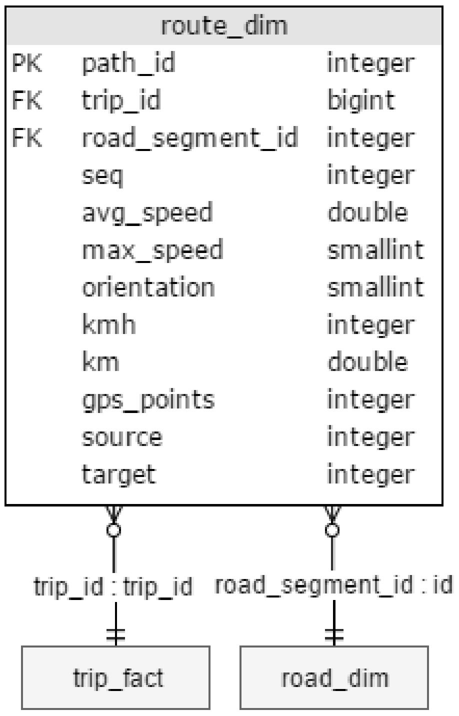
device_dim		
PK	device_id	integer
FK	device_type_id	smallint
	mac_address	varchar(40)
	class_of_device	varchar(6)
	device_uuid	uuid

Figure A.11: Device Dimension Table



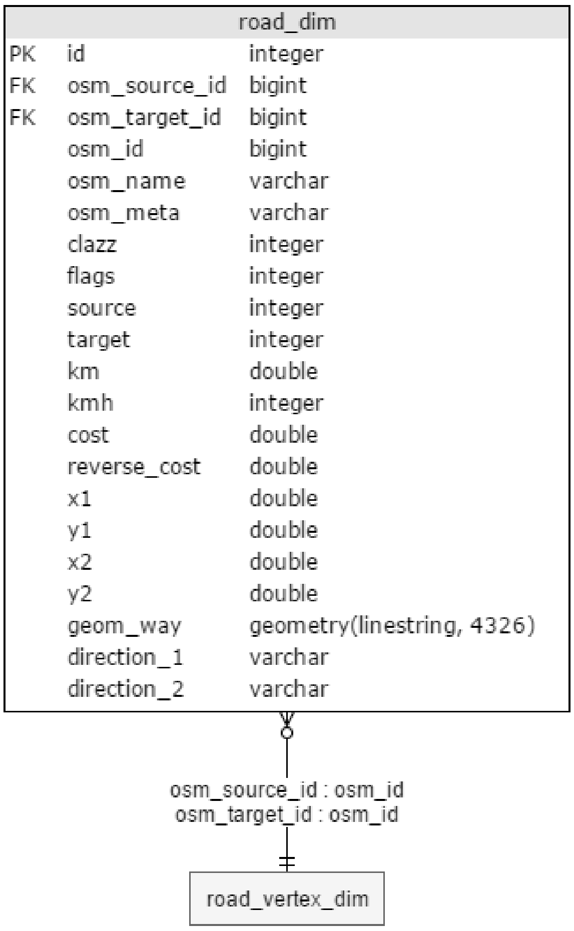
device_type_dim		
PK	device_type_id	smallint
	name	varchar

Figure A.12: Device Type Dimension Table



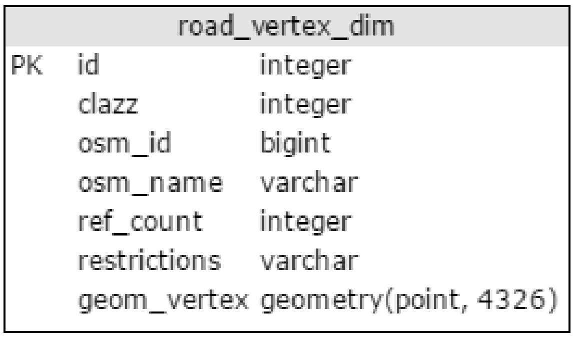
route_dim		
PK	path_id	integer
FK	trip_id	bigint
FK	road_segment_id	integer
	seq	integer
	avg_speed	double
	max_speed	smallint
	orientation	smallint
	kmh	integer
	km	double
	gps_points	integer
	source	integer
	target	integer

Figure A.13: Route Dimension Table



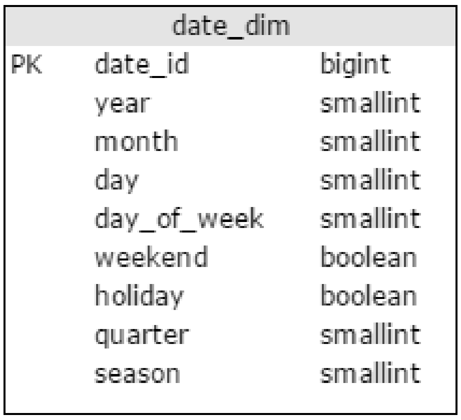
road_dim		
PK	id	integer
FK	osm_source_id	bigint
FK	osm_target_id	bigint
	osm_id	bigint
	osm_name	varchar
	osm_meta	varchar
	clazz	integer
	flags	integer
	source	integer
	target	integer
	km	double
	kmh	integer
	cost	double
	reverse_cost	double
	x1	double
	y1	double
	x2	double
	y2	double
	geom_way	geometry(linestring, 4326)
	direction_1	varchar
	direction_2	varchar

Figure A.14: Road Dimension Table



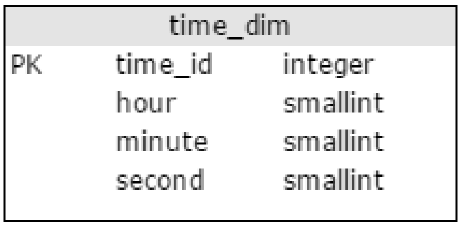
road_vertex_dim		
PK	id	integer
	clazz	integer
	osm_id	bigint
	osm_name	varchar
	ref_count	integer
	restrictions	varchar
	geom_vertex	geometry(point, 4326)

Figure A.15: Road Vertex Dimension Table



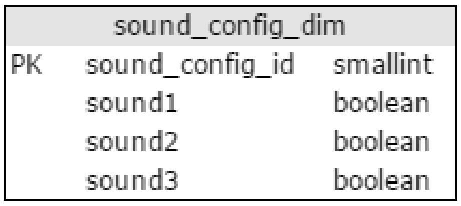
date_dim		
PK	date_id	bigint
	year	smallint
	month	smallint
	day	smallint
	day_of_week	smallint
	weekend	boolean
	holiday	boolean
	quarter	smallint
	season	smallint

Figure A.16: Date Dimension Table



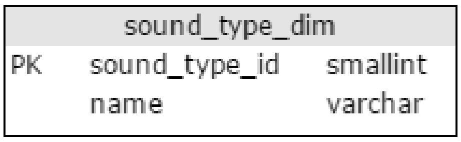
time_dim		
PK	time_id	integer
	hour	smallint
	minute	smallint
	second	smallint

Figure A.17: Time Dimension Table



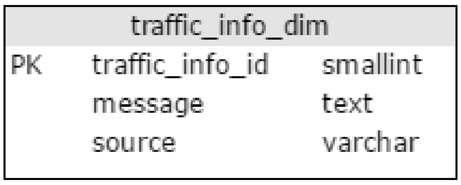
sound_config_dim		
PK	sound_config_id	smallint
	sound1	boolean
	sound2	boolean
	sound3	boolean

Figure A.18: Sound Configuration Dimension Table



sound_type_dim		
PK	sound_type_id	smallint
	name	varchar

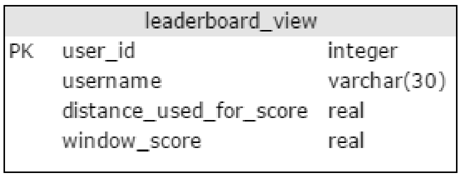
Figure A.19: Sound Type Dimension Table



traffic_info_dim		
PK	traffic_info_id	smallint
	message	text
	source	varchar

Figure A.20: Traffic Information Dimension Table

A.2.1 View Tables



leaderboard_view		
PK	user_id	integer
	username	varchar(30)
	distance_used_for_score	real
	window_score	real

Figure A.21: Leaderboard View Table