Data Analysis Platform for Taxi Fleets

João Temudo Vieira
Master in Data Science
Department of Computer Science
Faculty of Sciences of the University of Porto
2024

Data Analysis Platform for Taxi Fleets

João Temudo Vieira

Dissertation carried out as part of the
Master in Data Science

Department of Computer Science
Faculty of Sciences of the University of Porto
2024

Supervisor

Michel Celestino Paiva Ferreira, Assistant Professor, FCUP

Acknowledgements

Thank you to everyone who helped me in my scholarly journey, but especially:

To my parents, my brother and my sister in law.

To my remaining family.

To Diana, Hugo and Miguel.

To Geolink, and especially to Prof. Michel, Mr. Rui, Mrs. Isabel, André, Joaquim, Lucas, Ana, Mário and Marcus.

To my colleagues from MSc. Data Science, and in particular to João and Pedro, who inspire me to be better.

To my colleagues from BSc. Computer Science, who helped make university a more welcoming place.

And to all of my friends.

UNIVERSIDADE DO PORTO

Abstract

Faculdade de Ciências da Universidade do Porto
Departamento de Ciência de Computadores

MSc. Data Science

Data Analysis Platform for Taxi Fleets

by João VIEIRA

The adoption of a digital platform to complement the taxi industry represents a noticeable step forward towards challenging ride-hailing apps such as Uber and Bolt, which have become increasingly popular over the last decade and a half. The usage of such a platform can lead to a noticeable increase in management quality and strategic planning for taxi fleets, and client adherence to taxi services. Furthermore, the direct connection of taximeters to billing systems has been ordered by law by the Portuguese government, with a deadline of October 2024. In this thesis, we establish a platform for connecting taximeters to a mobile application with billing capabilities, along with collecting data from said taximeters for study and monitoring. The data collected from professional taxi drivers using this platform is thus studied to demonstrate some of the potential the collected data brings to change the industry. The results indicate a possible under-reporting of trips by taxi drivers to their managers and other partnered companies. We also establish several virtual dashboards that can showcase useful data for managers such as billing status of taxi trips, total earnings throughout a shift, and the portion of time a taxi spends on a trip during a shift.

Keywords: taxi, transport, data collection, docker, mobile application, dashboard

UNIVERSIDADE DO PORTO

Resumo

Faculdade de Ciências da Universidade do Porto
Departamento de Ciência de Computadores

Mestrado de Ciência de Dados

Plataforma de Análise de Dados para Frotas de Táxis

por João VIEIRA

A adoção de uma plataforma digital para apoiar a indústria de táxis representa um avanço significativo para o desafio de aplicações de transportes em veículos descaracterizados como a Uber e a Bolt, que se tornaram cada vez mais impopulares na última década e meia. Esta plataforma permite um grande aumento na qualidade de gestão e planeamento estratégico para frotas de táxis, e um aumento na adesão de clientes a serviços de taxi. Para além disso, um decreto-lei português ordena à ligação direta entre taxímetros e sistemas de faturação com um prazo limite de Outubro de 2024. Nesta dissertação, estabelecemos uma plataforma que permite ligar taxímetros diretamente a uma aplicação móvel com funcionalidades de faturação e colecionar dados oriundos destes taxímetros de modo a serem estudados e monitorizados. Os dados recolhidos de taxímetros instalados nos veículos de taxistas profissionais são estudados para demonstrar o potencial desta plataforma e da introdução destes novos dados à indústria de táxis. Os resultados obtidos indicam uma possível subnotificação das viagens efetuadas pelos condutores de táxis aos seus gestores e a empresas parceiras. Também estabelecemos vários painéis virtuais de controlo que exibem dados úteis para gestores, tal como o estado da faturação das viagens efetuadas por táxis, os ganhos totais de um turno, e a porção de tempo que um táxi passa em viagem durante um turno.

Palavras Chave: táxi, transportes, recolha de dados, docker, aplicação móvel, dashboard

Contents

List of Figures

List of Tables

Glossary

mD (km)	Mean distance (kilometers)
Dstd (km)	Distance standard deviation (kilometers)
mT (h:m:s)	Mean time (hours:minutes:seconds)
Tstd (h:m:s)	Time standard deviation (hours:minutes:seconds)
mF (€)	Mean fare (euros)
Fstd (€)	Fare standard deviation (euros)
mEKm	Mean euros per kilometer
EKmstd	Euros per kilometer standard deviation
AT/D	Average trip count per driver

Chapter 1

Introduction

Ride-hailing applications are a part of the new "sharing economy"[1], which is the practice of redistributing existing goods, such as homes in the case of Airbnb or vehicles in the case of Uber. They appear as heralds of the world entering a new digital age of greater and easier connection between both people and businesses, but their novelty also brings changes that must be (and are being) actively studied.

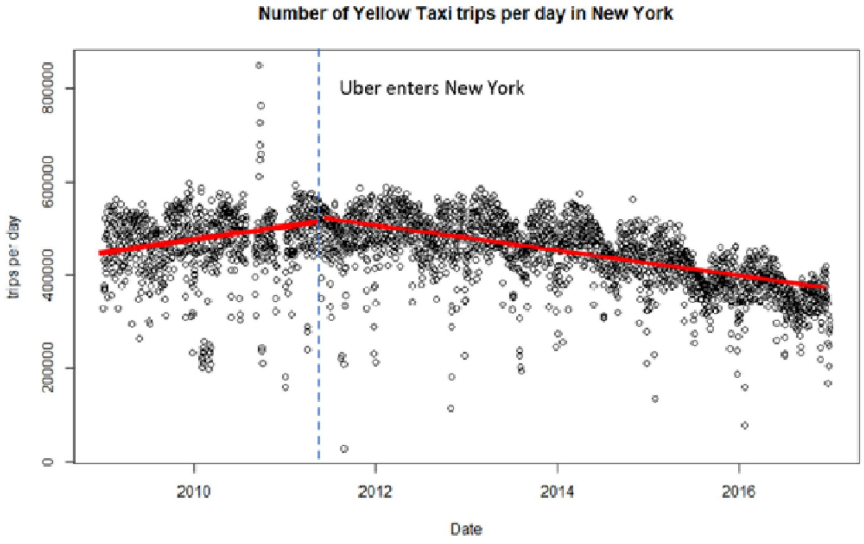


FIGURE 1.1: The impact of Uber on the taxi industry in New York[2]

These digital services have had a significant impact on the taxi market, beginning in 2009, when Uber first debuted its services[2], which marked a downturn in taxi trips every year since then. This decrease can be attributed to many factors: the advancements in mobile usage combined with the leverage of this previously untapped market[3][4] to allow it to reach a broad audience in a very short time; the focus on only the most lucrative areas due to a lack of regulation[5] allow its drivers to be more active on average; and the

lack of regulations due to being an entirely new economic practice allowed it to bypass limitations such as the temporal and economic cost of licensing drivers[6].

This impact is not limited to the taxi industry, however. The effects of these services underscore the urgent need for new legislation, as they contribute to traffic congestion[7] and pollution[8] on a significant scale. Beyond the structural and environmental issues, these digital ride-hailing applications have both necessitated[9] and attempted to influence[10] government action across the globe.

It's crucial to consider the potential risks Uber's operations pose to its consumers and drivers. The lack of well-informed consumer and producer protection laws in the "sharing economy" means that both the clients taking a ride and the drivers using vehicles are at risk of exploitation[11].

1.1 Purpose and objectives

This thesis aims to create a platform that not only strengthens the taxi industry by facilitating practices such as dynamic tariffs through greater availability of telemetric data but also allows for better management by supplying managers with detailed statistics on their drivers' behavior and performance.

This platform will also allow taxi drivers to continue their legal activities, as the new law passed on October 2023[12] requires taximeters to be directly connected to a driver's billing application to be allowed to perform their job. As such, the platform will have two main components.

The first component will connect the Taxi-Link Driver application, which functions paired with the Taxi-Link taxi-hailing application to connect clients to their drivers and leverage digital services to make billing and navigation easier for drivers, with taximeters of four brands: Digitax, Hale, Taxitronic, and Kienzle.

The second component is a server that collects telemetric data from the taximeters and prepares it for easy integration into multiple data analysis tools for an interested manager or supervisor to analyze. This server also includes Grafana dashboards with premade visualizations of important driver and fleet statistics, such as how much of their on-shift time is spent on a trip, how long their trips tend to be, how many trips are done daily, and more.

1.2 Thesis Structure

The remainder of this thesis is organized as follows: chapter 2 provides important background information to understand the influences behind this work; chapter 2.4 explains the advantages of the technologies chosen for this project; chapter 3 details the structure and implementation of the server-side components of this project; chapter 4 goes into the differences between each taximeter brand's software and the universal functionalities of the brands; chapter 5 analyses the results of both the Grafana implementations and a detailed statistical analysis of the data collected throughout this project's development; finally, chapter 6 includes the concluding remarks and potential future work.

Chapter 2

State of the Art

2.1 Ride-Hailing Services

Digital ride-hailing services have severely impacted the taxi industry, with Uber serving as both the flagship and pioneer of this new entry into the service industry for the past decade, and Gomez et al. (2021)[13] points out that these digital services are mostly being used by wealthy, educated, and car-prone individuals who would, before these services emerged, have otherwise called a taxi.

Several studies have been made to try to understand what the root cause of this impact *is*. Pepić (2018)[14] posits that one of Uber's primary advantages lies in its drivers not being provided with vehicles, which allows the company to curb prices by only needing to maintain their digital infrastructure; Cramer et al. (2016)[15] puts forward that dynamic pricing and lack of zone restriction for drivers are the most critical reasons behind Uber's high adoption rate. Frechette et al. (2019)[16] quotes the knowledge of where a client is at any point in time as one of Uber's most noticeable advantages over taxi services. Dudley et al. (2017)[17] emphasize how Uber exists both as a disruptor of an existing market and is even capable of taking advantage of protests from taxi drivers to increase its own public standing and consumption.

Another notable advantage of these new services over established taxi services lies in their relationship with legislation. Väyryren et al. (2020)[18] serves to highlight how, while laws can have good intentions and try to aid the taxi industry through *de*regulation, they can sometimes undermine it by creating ambiguity in how the industry may proceed with new legal changes; Cramer et al. (2016)[15] and Berger et al. (2018)[6], on the other

hand, take into account the advantages Uber has by having registered itself as an IT company, specifically in how it is allowed to function both over space by not being restricted in what drivers are in its area and in regards to its employees not needing any licensing, which has both an associated cost and a legal limit on emitted licenses.

Harding et al. (2016)[19] goes as far as to propose that Uber has challenged two of the three main assumptions behind taxi regulation proposed in Cooper et al. (2016)[20]'s QQE model, those being *Quality* and *Quantity* control, and largely mitigated the third, *Economic* control; Thelen (2018)[21] suggests that the impact Uber is "allowed" to have is limited by its environment: the USA, Germany, and Sweden showed vastly different outcomes with the introduction of Uber, both due to the actions of the taxi associations, but also the measures undertaken by regulatory bodies. Martins (2019)[22] details the actions taken in Portugal, emphasizing the focus on regulating Uber and other similar companies over adapting existing taxi regulations to handle this change in the transportation industry.

Beyond the impact Uber had up until 2020, Gu et al. (2024)[23] state that the Covid-19 pandemic and other future events of similar, worldwide impact might generate "winner takes all" scenarios if local authorities do not take care to prevent monopolies.

2.2 Dynamic Fare Pricing

Dynamic pricing, that is, the practice of establishing different prices based on factors such as demand, location, and time of day, has been established to be one of the cornerstones behind Uber's success, both by Guo et al. (2017) [24], and by Yan et al. (2019)[25].

Since Uber expanded into the worldwide market in 2011, several studies have been conducted to determine how the taxi industry could adopt such a system. For this purpose, several models have been developed, such as Qian et al. (2017)[26]'s proposed time of day model. Xu et al. (2022)[27]'s proposed model also takes user behavior into account, rewarding responsible ordering of services and punishing last-minute cancellations on top of considering service demand.

Jin et al. (2019)[28], on the other hand, proposes a system that, instead of taking the time of day into account or a user's behavioral quality, uses supply and demand as a baseline for how taxi pricing should be decided, as illustrated by figure 2.1.

The study of how this system could best be developed by Egan et al. (2018)[29], and what data would need to be sourced for it to function, highlights how much information would be necessary at any given moment to apply dynamic pricing on an individual

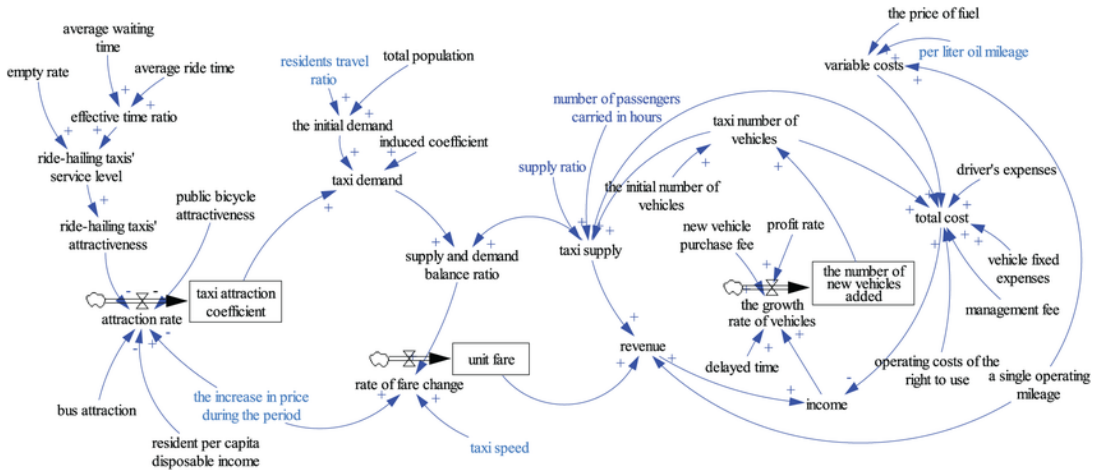


FIGURE 2.1: Supply and demand model proposed by Jin et al. (2019)[28]

taxi's scale, both in terms of a taxi's physical status, and its driver's wants and needs. The adoption of dynamic pricing by fully electric taxi fleets, and the specialized data that would need to be provided to make it function in a manner that most benefits from its fuel source's nature, has also been a subject of study, as seen in Maljkovic et al. (2023)[30]'s work.

This adoption is not necessarily an easy one from a societal standpoint either, as studied by Tervo et al. (2022)[31], which highlights 10 main reasons for the non-adoption of this system; said study mentions both emotional reasons, such as a desire to uphold traditional taxi values, and logistical reasons, such as certified taximeters' needs for parameters to be set in advance. On the other hand, Guo et al. (2022)[32] brings up the need for a transparent system in order to ease customers into the usage of such a system, lest they grow distrustful of a system that they feel is taking advantage of them.

2.3 Digital Transformation

The digital transformation of the service industry, connected to Industry 4.0 as detailed by Pandya et al. (2023)[33], is a critical component of adopting dynamic fare pricing and of this thesis; beyond that, it is also an important factor in keeping up with the advancement of the rest of the service industry, as detailed by Esther et al. (2024)[34].

Of particular interest to this thesis is the impact of incorporating this digital factor into the taxi industry. As suggested by Shr et al. (2024)[35], adopting digital platforms by taxi drivers can greatly increase taxi profits and the quality of service for the clients. Furthermore, Harding et al. (2016)[19] propose that adopting mobile applications for taxi

usage would strengthen the taxi industry and make it more competitive with services such as Uber, a theory that Akbulaev (2020)[36] also arrives at, with the added note that, with the existence of mobile application services for taxis, the ones that do *not* adopt such services are likely to suffer from decreased clientele.

Integrating digital applications into the taxi industry also allows for easier management of taxi fleets: Brodeur et al. (2018)[37] details how, when compared to digitalized platforms such as Uber and Lyft, the entities in charge of taxi fleet management were incapable of taking advantage of the increase in clientele due to lacking a way to know *when* to mobilize more drivers to take advantage of favorable conditions. Other studies, such as Zhang et al. (2015)[38] and Ozeki et al. (2023)[39], detail how obtaining GPS data can aid in improving taxi fleets' efficiency by creating models of when and where there will be more activity so taxis can prepare in advance. On the other hand, Mae et al. (2017)[40] propose that recognizing why problems occur in a taxi service is critical for the continuation of the industry, which a digital platform would facilitate.

To ensure that the taxi industry does not die out in Portugal but maintains its full legal and institutional legitimacy, this thesis will focus on integrating the Taxi-Link Driver app with taximeters to enable smarter, more efficient management of the over 3000 taxis working in cooperation with it and to create an opportunity for implementing dynamic pricing in Portugal's taxi industry.

2.4 Technologies used

2.4.1 Mobile application

The mobile application modules developed had little to no options regarding the technologies used due to the need to integrate them into an already existing application. As such, Android Studio was used for the module development, with Kotlin as the programming language for the logic components and XML files for the visual components.

Android Studio is an IDE that comes pre-packaged with several useful functionalities and has been an industry staple for a long time, as noted by both Dimarzio et al. (2016)[41] and Wolfson et al. (2013)[42]. It has an integrated mobile simulator, offers code suggestions, hyperlinks classes and functions to their implementations and objects to their declarations, and has many other quality-of-life features common amongst other

IDEs, such as refactoring a class, function or object name and searching across multiple files for text, all optimized for mobile development.

Kotlin is an open-source language that runs on the Java Virtual Machine, enabling it to function on any machine Java can run on. It is also compatible with Java libraries and frameworks. Kotlin has several differences and notable advantages over Java. Despite Java's overall popularity, it has become a strong competitor, especially in Android development.

A key difference between Kotlin and Java, which was particularly important for this project, is how each language handles null safety. Whereas Java does not have much in the way of natively handling null pointers (beyond if statements for checking if they *are* null), Kotlin is capable of handling them in various ways. Variables can be declared as nullable, should they be given a type, and the usage of suffixes when such a variable is called can change its behavior to be fully null safe (with even access to properties of properties not causing exceptions) or to treat the variable as non-nullable.

Another important difference is found in each language's syntax, as Kotlin's is very concise and descriptive, which limits how much boilerplate code is necessary for a given project. This more interpretable syntax is key in making projects in Kotlin more readable, allowing for an easy start to developing the mobile application modules.

Kotlin code can fully integrate Java code into its functionality, which is critical for developing this project because some of the necessary proprietary APIs are coded in Java.

2.4.2 Container Software

To more easily manage the several applications that need to be hosted on a company-owned server, Merkel et al. (2014)[43] suggest that container management software allows for the separation of all the services without the risk of a problem in one affecting the others, along with an easy setup of the environment for whatever new applications one may need. Furthermore, using containers facilitates the port-forwarding of different applications to different ports more easily, which is very important when it comes to accessing different applications on the same machine. In this project, three ports were forwarded to:

- Manage and monitor the various containers for the server-side software;

- Allow for internet access to the web service that handles data collection and processing;
- Access the data monitoring and statistics dashboards.

The container software management chosen for this task was Docker, as it contains every necessary feature required to develop the various server-side applications. Thanks to the strong community presence and regular usage for server-hosted applications, there were several options for pre-made docker images for this project's data collection *and* data analysis needs. Docker is also one of, if not *the* most popular software of its type, meaning that many other applications have integrations to facilitate its usage and compatibility. For example, Visual Studio Code, the editor used to develop and manage server-side applications, has several extensions to facilitate Docker usage.

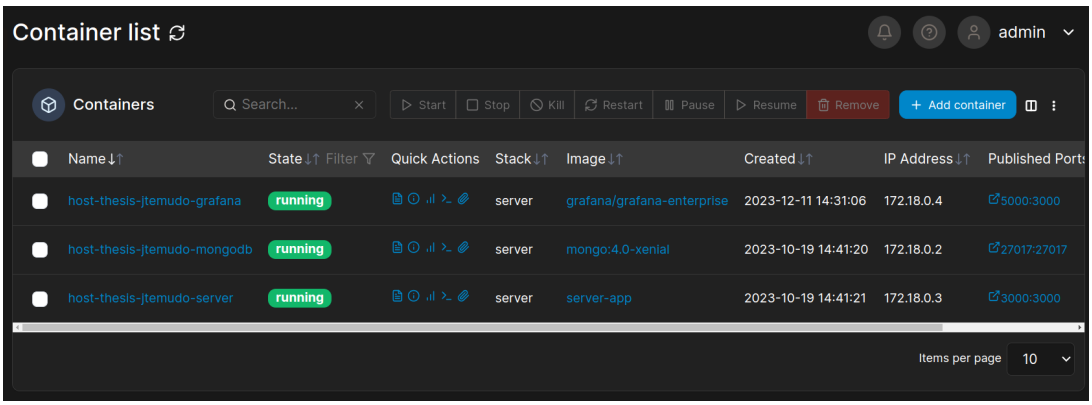


FIGURE 2.2: Portainer, a platform for managing docker containers

Docker also has tools that make developers' lives easier, such as Portainer, shown in figure 2.2, which allows for monitoring each container and provides an intuitive GUI that permits much easier configuration and testing of new configurations and new containers. Furthermore, Docker's popularity has also led to the publication of many internet articles and blog posts that explain the basics of how Docker works, what a new user needs to understand, and how to leverage the resources available to each container to their maximum potential.

2.4.3 Web Service

Because they were developed from the ground up, the web service that handles data collection and processing and the database that stores all records obtained through the

mobile application were carefully handpicked to facilitate usage, maintenance, and scalability.

The web service will be the most critical element in handling the data collected by the mobile app. It will preprocess the received data, insert it into the database, and send responses to the mobile app. It will also need to perform these tasks quickly, seeing as the number of taxi trips at a given moment can ”spike” at certain intervals of the day, heretofore referred to as ”peak hour”.

The chosen environment for developing this web service was node.js, an open-source and cross-platform JavaScript runtime environment that can run on any platform and, most importantly, handles requests asynchronously. Handling requests asynchronously means the higher activity during peak hours will not significantly affect the service, which is important due to the nature of taxi services, which can experience spikes. This is very important as not only does the data collection step rely on the stability of the service platform, but so does the mobile app, as is explained in section 4.

Node.js also features very easy methods of separating and integrating APIs, as an API file can be registered using two code lines in the main application file for a node application. Using Express, a framework for developing Rest APIs in node.js described in (2016)[44], the development of new APIs in incremental steps can be easily accomodated. Furthermore, this technology is also particularly suited for working in teams, as it allows for parallel development of different files with no interference with the functionalities defined in others.

```
const express = require('express');
const app = express();

//routes for handling trip collection entries
const tripRoutes = require('./routes/tripRoutes.js')
app.use('/api/trip', tripRoutes)

//routes for handling odometer entries
const odoRoutes = require('./routes/odoRoutes');
app.use('/api/odo', odoRoutes)

//routes for handling complete state instances
```

```
const stateRoutes = require('./routes/stateRoutes')
app.use('/api/state', stateRoutes)

//routes for data relating to global taxi data
const taxiRoutes = require('./routes/taxiRoutes.js')
app.use('/api/taxi', taxiRoutes)

//routes for configuring and retrieving hale licenses
const haleRoutes = require('./routes/haleRoutes.js')
app.use('/api/hale', haleRoutes)

// Start the server
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

2.4.4 Data Storage

The chosen software for the application's data storage is MongoDB, a NoSQL DBMS that can store irregularly formatted data and is particularly adept at these tasks [45]. This is important for the project's development, as data entries from the many taximeter brands integrated with the server for data collection will vary depending on the available data. For example, while the Digitax integrated taximeters can detect which tariff is currently active during a trip, the Hale taximeters cannot, necessitating either using a "filler" value or, thanks to MongoDB, removing the field from the entries that come from Hale taximeters. MongoDB also offers advantages beyond the capability to store the irregularly formatted data generated by the different hardware used to grow the database.

MongoDB's storage model is document-oriented, meaning it creates a file for every entry in a table and uses JSON formatting to store the relevant details. This makes the data readily available in many programming languages, as the attribute-value pair structure is easily interpretable and grants it a very rich API, according to Hecht et al. (2011) [46].

Thanks to the ability to set fields to serve as indexes even after database creation, querying through MongoDB can be improved over time by setting each collection's most

frequently queried fields to serve as indexes. Cornelia et al. (2022)[47] note that, when coupled with its powerful querying language, despite its learning curve, MongoDB allows for complex and efficient querying, even when the database has accrued months' worth of data.

Regarding scalability and data integrity, MongoDB is a tool that presents many options. The ability to enact sharding, that is, the process of storing one database across several machines, allows for both scalability, as more machines can be added to the system to increase storage space, and mitigating the impact of single machine hardware failures, as the data stored in separate machines is unaffected by local issues. Furthermore, MongoDB can utilize data replication across multiple machines to decrease the chances of losing data due to a single machine's hardware failure. Makris et al. (2019)[48], Makris et al. (2021)[49], and Agarwal et al. (2016)[50] note that MongoDB also possesses geographical data storage capabilities that are, if not as good, at least comparable to leading DBMS used in the industry. This is useful for preparing for potential future work and enabling some of the data monitoring done throughout this project.

Figure 2.3 displays a simple schema of how the project's server infrastructure is planned: the data is collected on a mobile application, then sent to the server, which processes the received data and stores it in the appropriate database.

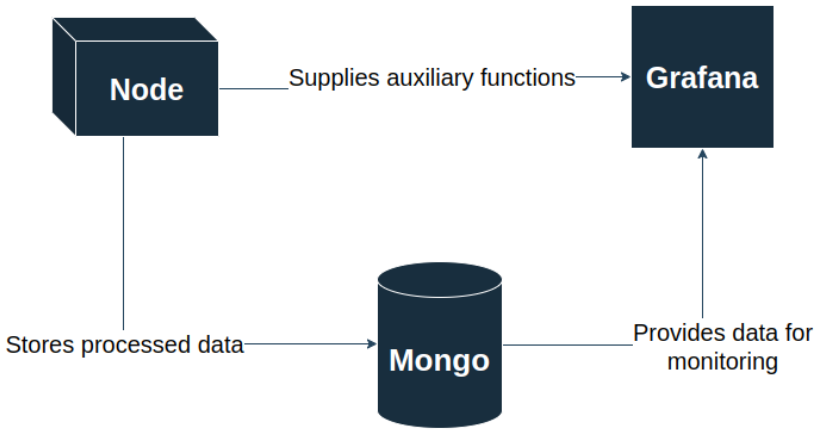


FIGURE 2.3: Organization of the server virtual machines and how they interact.

Chapter 3

Server Structure

This chapter describes the architecture and structure of the server that receives, processes and stores the information gathered from the mobile application, described in 4.

3.1 Hardware Capabilities

The server is hosted within a single virtual machine, and needs to have the capabilities to handle the load generated not only from running each virtual machine specified in subsection 3.2, but also to expediently process the frequent communications between the various drivers' applications and the rest API defined in section 3.3, along with the resulting CRUD operations performed on the MongoDB database described in section 3.4.

To handle this burden, the virtual machine was provided with dedicated resources, namely disk space, RAM, and CPU cores/threads. Since each core possesses two threads, the number of allocated threads is always double the number of allocated cores.

Machine	RAM	Disk Space	CPU Cores	CPU Threads
Real	62.36GB	100GB	24	48
Virtual	4GB	50GB	4	8

TABLE 3.1: Real vs Virtual Machine Specs

The allocated resources are based on the needs of the server, database, and monitoring software. As explained in sections 3.3, 3.4 and 5.1, most operations being performed are CRUD operations and relatively simple arithmetic, meaning that RAM is allocated and released at a very fast pace. Furthermore, since each database entry is very light, and the operations performed on them are simple, high processing power is unnecessary, freeing the CPU for usage in other applications running on the physical machine. The database,

on the other hand, necessitates a high amount of disk space, leading to half of all available disk space being allocated to the VM.

3.2 Virtual Machines

Within the virtual machine described in section 3.1, 3 docker containers are running at all times:

1. *Node*, which contains the Rest API documented in section 3.3 and serves as the data collector for the whole project;
2. *Mongo*, named after the MongoDB structure described in section 3.4, stores all of the data used and processed for this thesis;
3. *Grafana*, named after the monitoring tool *Grafana*, which is explained in section 5.1, allows for the monitoring of taxis in real time.

Of these containers, both *Node* and *Grafana* are connected to *Mongo*, as they rely on its data storage to perform their functions. *Mongo* is not meant to be interacted with directly, except for performing data analysis on the data contained within with tools such as *Python*'s *Pandas* and *NumPy* and storing data from *Node*.

3.3 Node.js API

The server API is split into several modules, each related to a different type of collected data. The size of each segment is dependent not only on how often the data is interacted with but also on how important it is to the proper functioning of this work.

3.3.1 Odometer Module

The odometer module, which uses the prefix */odo* for its endpoints, handles entries related to "snapshots" of a taxi's odometer while it performs a trip. This module receives most of the data entries sent by a taxi and then transports them into the database both directly and after transforming them for other modules. Since this module captures snapshots, it can safely store the geographical coordinates of the taxi at each point.

Data collected for this module is very granular, as it tends to be collected at regular, albeit short, intervals. This module's most frequently used endpoint is */odo/group*, which

receives a series of data points from a single taximeter and processes them all at once. The processed result is then stored in two data models, described in section 3.4: *steps*, which record the geographical coordinates of a taxi throughout time, and *state instances*, which record the duration of a state, the distance traveled according to the taximeter during said state, and, if the taximeter reports a trip, the tariff and supplements' costs.

FIGURE 3.1: An application of the module that allows for easy visualization of the results obtained by a taxi fleet during January of 2024.

An example of the usage of state instances can be seen in figure 3.1, which reflects the trips made in January 2024 that were recorded using the mobile application by a single company and that were over 500 meters in distance. Each point in the graphs represents the total value in euros obtained each day of the month, or, in the rightmost graph, the total trips done on said day.

This module allows taxi fleets to track their earnings daily, narrow down the results to one driver, and even check their efficiency, as shown in section 5.1. Furthermore, since these trips have both the distance recorded by the taximeter *and* the geographical coordinates traveled during the trip, there is now a greater simplicity in detecting fare tampering when compared to before the implementation of this module.

3.3.2 Trips Module

The trips module, which uses the prefix */trips*, stores the trips each taxi reports as recorded by the taximeter's software. This module is made to interact specifically with the *Digitax* brand of taximeters, explained in subsection 4.2.1, as its powerful API allows for capturing trips that would otherwise go unreported by employees. This is then monitored on a dashboard, shown in figure 3.2, dedicated to tracking which trips were billed, which were not reported, and each bill's trip.

The development of this module was brought about due to a need to check for the aforementioned unreported bills. A taxi company was having issues with certain drivers turning off the application, which could have led to losing access to European Union

Início de Viagem	Final de Viagem ↓	Distância Percorrida	Tarifa	Suplementos	Faturada
07-08-24 16:38	07-08-24 16:44	3.85 km	€5.05	€0.80	Sim
07-08-24 15:43	07-08-24 15:49	3.66 km	€4.95	€0.00	Não
07-08-24 14:42	07-08-24 14:49	4.06 km	€5.25	€0.80	Sim
07-08-24 14:20	07-08-24 14:24	2.01 km	€4.15	€0.00	Não
07-08-24 12:53	07-08-24 12:57	1.52 km	€3.95	€0.00	Sim
07-08-24 12:26	07-08-24 12:32	1.32 km	€4.25	€0.80	Sim

FIGURE 3.2: An example of taxi trips as seen in the dashboard made available to companies using the trips API.

funds, and thus asked for the development of a module capable of assisting in detecting these unreported trips so they could be handled in-house.

The data collected for this module is entirely new to the taxi company ecosystem, and the data that can now be incorporated into taxi companies' functioning opens up new possibilities. The ability to detect unreported taximeter trips, to know the distance undergone during, and the monetary compensation obtained from these trips, along with their start and finish dates, allows for much easier detection of fraudulent returns for the purposes of both the taxi company and governmental purposes than ever before.

3.3.3 Hale Module

The Hale module is designed to handle the unique requirements that the brand's taximeters bring to bear. Namely, to access their data collection features, a license must be requested and paid for at regular intervals, with each taximeter requiring its license. This license must be requested from Hale's own server API and is then stored within Geolink's database.

While there are many different licenses, each with its feature packages, the one that was used for this work was a novelty license that included two packages: the Basics package, which includes features such as the state of the current trip, if any, the distance traveled within it, and the current fare and supplements, and the Totalizers package,

3.4 MongoDB Storage

The MongoDB collections created for this project were reworked several times, as initial methodologies required changing due to their non-scalable nature. As such, each module shown above enters data into different collections, which have varying internal structures.

The advantages of MongoDB were crucial in this segment, as data from older entries that followed different base schemas was not wasted nor required any extra work to adapt upon changing the server's data collection operations. This allows for easier analysis of the data and allows different taximeter brands to send data that is only available from that brand.

For the purposes of the analysis done in section 5.2, however, the schema for these collections is effectively the one shown in figure 3.3.

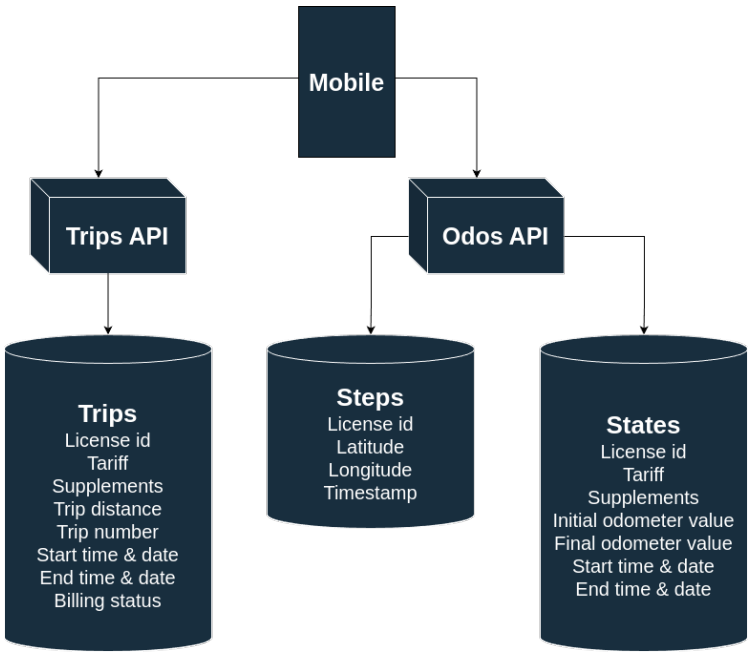


FIGURE 3.3: How and where the server stores data originating from the mobile application

3.4.1 Collected Data

By August 14, 2024, 394,995 state instances had been collected since the implementation of the Odometer module defined in subsection 3.3.1 in early November 2023. A state instance is defined by any interval of time in which a taximeter changes state between free, occupied, and in payment. These entries always carry the taximeter ID, the distance the taxi traveled at the start and end of the state instance, both in total and while hired, the start and end date of that instance, and the state. Occupied instances in which a taxi is undergoing a paid trip also include the tariff, the fare, and the supplements for that trip.

The trips collection, being a more restricted addition due to only being available for Digitax taximeters, had only collected 54,928 entries by the aforementioned date. This can be attributed to its ability to collect historic trips from Digitax taximeters, which makes it

capable of collecting 1/8th of the number of *instances* in the much smaller interval starting in early June 2024. However, due to the Hale taximeters' entries, along with payment and free instances, there is a great disconnect between the total entries collected for each collection.

The steps collection, first established in late March 2024, has collected 43,819,055 entries since its inception. It is a successor to the odos collection, which had collected 531,388 entries before its discontinuation. A step entry contains the ID of the taxi that it pertains to, a timestamp, and the taxi's GPS coordinates at that time. An odo entry contained, beyond the data in step, the total and hired distance traveled by taxi, the current state of the taximeter, and the accumulated distance the taxi had undergone since it started its current trip, if any.

The odos collection was discontinued due to the introduction of the state instance, which stores the same data minus the coordinates in a much more compact and interpretable format.

3.4.2 Regular Data Usage

The way the data is interacted with on a regular basis is almost entirely limited to insertions, updates, and database reads. As mentioned in 3.3.1 and 3.3.2, there is an almost constant stream of data being received and inserted into the database.

State instances, as described in 3.3.1, record the data of a continuous stay in the same state and are updated every time a new entry is sent to the odos module. This means that, although there is a constant flow of data entries coming into the database, the data entries themselves are lightweight, and thanks to having set the time fields as indexed fields, the read and update operations can be performed swiftly and efficiently.

Chapter 4

Mobile Application

This chapter details the behavior of the taxi driver application that serves to collect data, which is then sent to the server described in section 3.3. In it, the different methods through which the app connects to the taximeter, what data is collected, the limitations of each brand, and encountered difficulties are explained.

4.1 Implementation Methodology

Thanks to Kotlin's Bluetooth API, certain functions can be implemented for all taximeter brands and can be expected to function without issue. Namely, there are functions for searching for the desired device, defining behavior for disconnection events, and other such utilities.

By utilizing a factory method pattern, the differences between each taximeter brand are taken into account and handled to provide the rest of the application with the data it needs to function properly. As such, although the internal behavior of each brand can vary greatly due to the methodology specifics, a baseline for observable behavior is expected from the rest of the application.

The factory method pattern has several advantages[51], with the most relevant ones for this project being how it allows for the code that regulates the connection object's declaration to be far more readable due to creating a function that, when provided with the target brand, calls the correct implementation of the connection interface. This creates a clean separation of the created object's usage and creation logic and an easy way to expand connection capabilities to add more taximeter brands to the project in the future.

The important features that are implemented across all taximeter brands for this project are described in the following subsections.

4.1.1 Concurrence

Mobile applications use thread pools, with the main thread being the only one allowed to interact with the UI, which it governs. Due to blocking said UI, the main thread is unable to use asynchronous functions. As such, the need to process the data coming from the taximeter in its own thread is raised. To address this need, the implementation of the connection handling classes extends the Thread class, allowing for the data processing, Bluetooth connection, and server communication operations to be handled in the background.

Beyond this need for concurrent processing, there also comes a need for concurrent data sharing. Since the taximeter data needs to be accessible throughout multiple threads, MutableLiveData objects from Android's standard development API were used to store the data to make it accessible from all threads. These objects allow multiple threads to post data concurrently by calling the PostData() method, which handles these calls by queuing them and executing each in FIFO order. Observers can also be set over the MutableLiveData objects, which use the caller's lifecycle (i.e. a Fragment's lifecycle if this observable should only exist while on a certain page) to determine their active state. When a MutableLiveData object changes its value, it notifies all Observers set over it, which then read the new data and process it in the main thread, allowing it to interact with the UI.

4.1.2 Boot-up connection

When the mobile application is launched, it will automatically try to establish a connection to the selected brand's taximeter. This is done by implementing the connection function defined in the application's bespoke device interface and calling it at the start of the thread's lifetime. This is done without any need for the user's input, increasing the application's ease of use and, thus, adhesion by its users.

The specifics of the connection can vary across brands, but all of the implementations follow the same basic formula:

1. Verify if the taximeter is already connected;
2. If not connected to a device, search throughout all available paired Bluetooth devices and try to find one of the correct brand;
3. If no device is found, wait for a short period of time and return to step 2;
4. If one such device is found, connect to it and store its MAC address on the server so the search is faster in future application launches;
5. Once connected to a device, the data collection protocols of each device are initiated, and the loop is ended.

Once the connection step has been concluded, the data collection process is launched, and the server receives a signal that the taxi is ready to begin receiving services, both off the street and from the central.

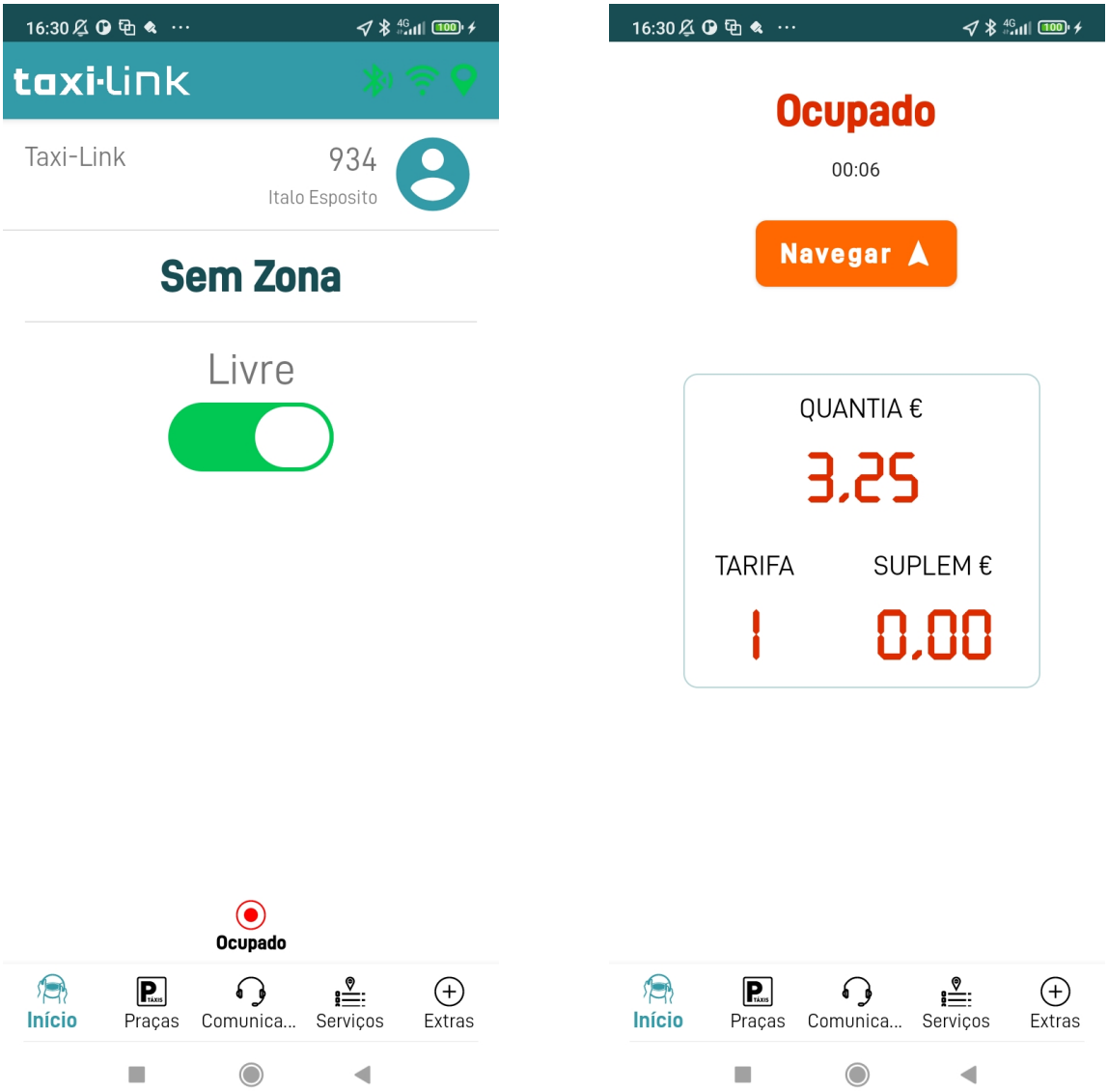
4.1.3 Automatic reconnection

When the connection to a taximeter is lost, the application will automatically attempt to reconnect once Kotlin's Bluetooth API recognizes the disconnection. This is done by making the data collection loop and the connection loop the same, with a verification of the connection state variable at the start of each iteration, determining the expected behavior.

By establishing this loop in such a manner and taking advantage of null safety to ensure a disconnection mid-loop does not crash the mobile application, there is also a guarantee that data collection will not interfere with the connection step and that disconnections will only pause data collection instead of halting it fully and necessitating a full restart.

4.1.4 Automatic state management

When a taximeter's state changes (or, in the case of Taxitronic taximeters, changes specifically to payment), the application is notified automatically. When this state changes, the application not only shifts to different Fragments but also notifies the server of the taxi's new state, meaning that it won't attribute new services to that taxi. These state changes can be seen in figure 4.1.



(A) Main fragment for when not on a trip

(B) Trip fragment once a trip starts with most brands

FIGURE 4.1: The different UI states the application goes through depending on the taximeter state

This state change is made using MutableLiveData and Observers, which allow for real-time, concurrent updates of the application's UI and functionality. The code shown below contains a snippet of the workings within the screen in figure 4.2a, wherein once the taximeter goes from being in a busy or paying state to a free state, the UI automatically shifts back to the screen in figure 4.1a.

```
class ServiceOngoingFragment : Fragment(), ... {
    override fun onCreateView(...): View {
    TaximeterRepository.taximeterBusy.observe(
```

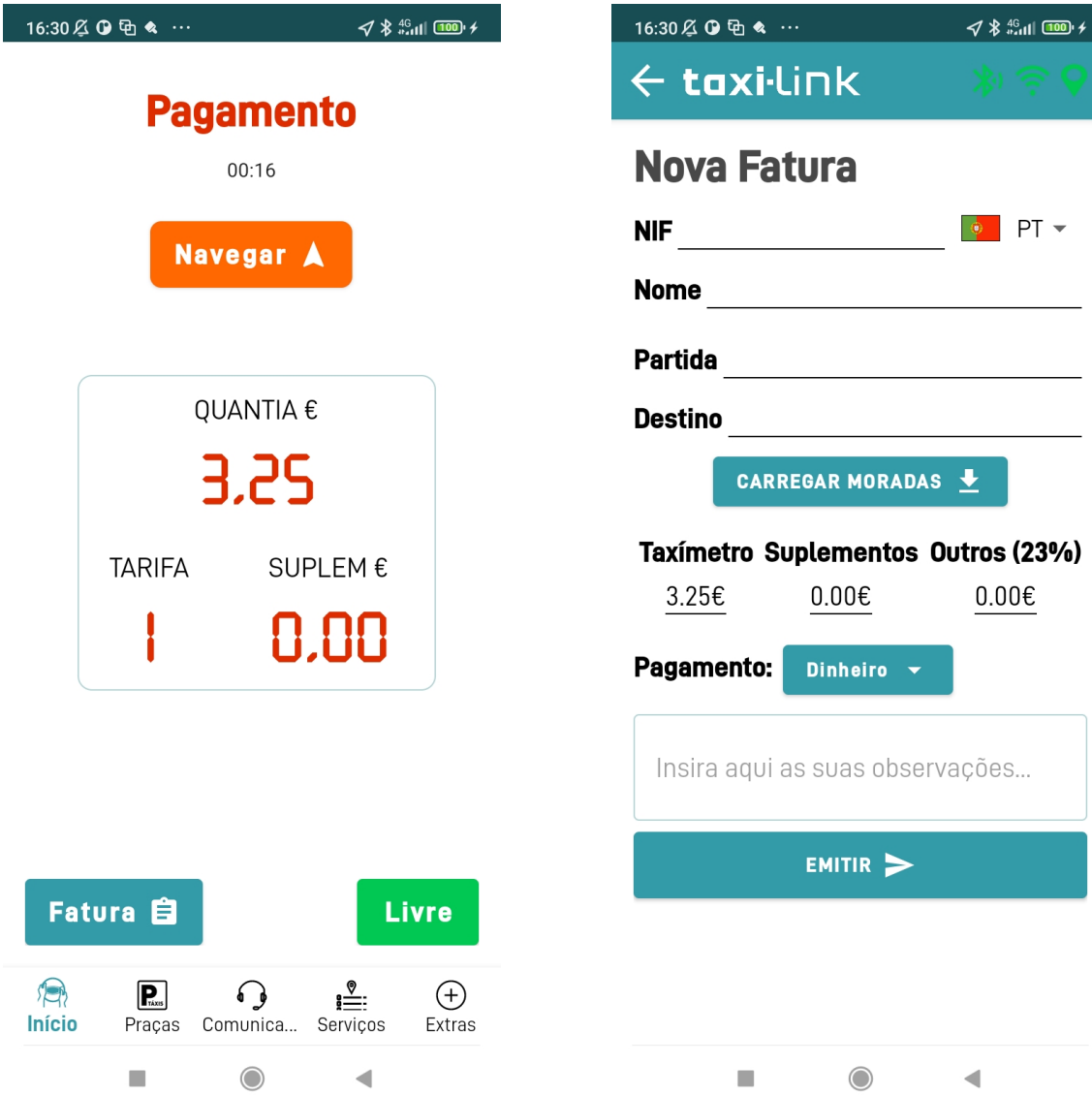
```
        this.viewLifecycleOwner, stateObserver)
    }

    private val stateObserver = object : Observer<Boolean>{
        override fun onChanged(busy: Boolean?) {
            if (busy == false && TaxiState.state.value in arrayOf(
                    TaxiStatusType.FREE,
                    TaxiStatusType.PARKED,
                    TaxiStatusType.PAUSE)) {
                val action = ServiceOngoingFragmentDirections.
                    actionServiceOngoingFragmentToMainFragment()
                view?.findNavController()?.navigate(action)
            }
        }
    }
}
```

4.1.5 Automatic billing data

As explained in section 1.1, there is now a demand for taximeter data to be transmitted directly to billing applications without the taxi driver's input. Taxi-link, a leading force in the market, is one of the first applications to do all of these processes self-contained, without needing third-party billing or, in most cases, taximeter handling applications.

This process is done using the same method as the state updates described in subsection 4.1.4, but instead of updating the UI, the values for the latest trip are loaded into the billing information instead, as shown in figure 4.2.



(A) Trip fragment in a payment state

(B) Fragment for billing a trip

FIGURE 4.2: The different UI states the application goes through depending on the taximeter state

This billing data is almost always non-interactive, meaning that a taxi driver cannot change the fields filled in by the application. After the billing process is concluded, the application allows for the emission of new bills by the driver with custom values, as is the case for contract trips, which show their value on the taximeter as 0.00€ and thus must have their value manually inputted.

4.2 Taximeter Brand Specifics

Each taximeter has its own implementation methodology, as there isn't a unified communication protocol, be it in the data they allow to be collected or in how they transfer it.

At the end of each brand's subsection, a code snippet containing their connection/data collection cycle is shown.

4.2.1 Digitax Implementation

The Digitax brand taximeters are very simple to integrate, as they come with their own pre-written API developed by their manufacturer company in Java. Due to Kotlin's ability, described in section 2.4.1, to integrate native Java code into itself, this API was immediately usable without any further alterations regarding its functions' behavior.

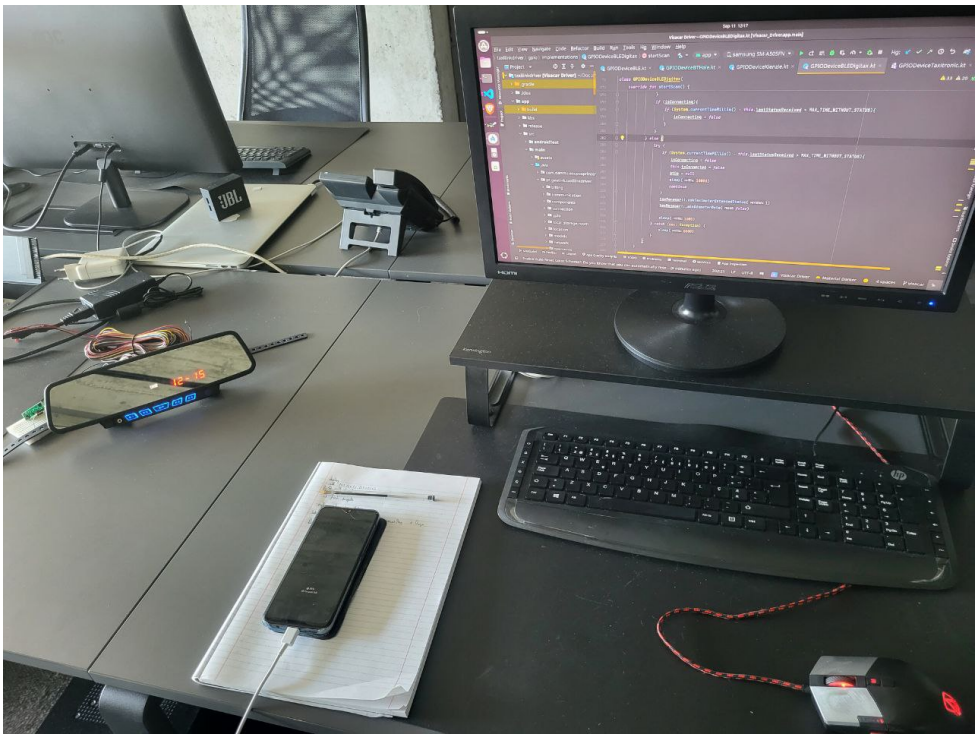


FIGURE 4.3: Workplace environment for Digitax-related work

Working on integrating Digitax taximeters has multiple requirements on the hardware side, however, as they do not come with any auxiliary tools for testing. As such, the workplace environment was similar to the one shown in figure 4.3, with a computer for code development, a phone to test the connection capabilities, a taximeter to test the values, and a Raspberry Pi computer to send IO signals through a breadboard to simulate the

signals coming from a moving car. This setup required setting up a Raspberry Pi and coding it through remote access, checking the Digitax manual for the hardware details on which pins are related to the impulse counters, and creating a breadboard circuit that would simulate these signals with safe values.

Digitax taximeters use the provided API to establish their connection, streamlining the search process down to calling a single function and then handling the callback to store the Bluetooth-paired device object returned with it, with the assurance that this device is a Digitax taximeter. This object, which extends Java's BluetoothDevice class from its native API, is then stored so the global functionalities defined in section 4.1 can be properly implemented.

Once the application has established a connection to the taximeter, it sends a signal to the device to define behavior that is only possible thanks to the brand's API, namely, to send some of its internal records to the application at regular intervals without the application needing to do so in a loop. These records include whether the taximeter is on a trip, in payment, or free, as well as the current fare, supplements, and tariff, the total distance traveled, and the distance traveled for the current trip, if any. This data is immediately sent to the server once received, in JSON format, after being encoded to be more easily stored.

```
override fun run(){
    while (true){
        if (!isRunning){
            sleep(30000)
        }
        //enter connection cycle
        if(!this.txConnected){
            if(btManagerDigitax?.IsScanning() == true){
                sleep(1000)
                continue
            }
            val lastScan = btManagerDigitax?.LastScansionResultGet(true)
            if(!lastScan.isNullOrEmpty() && !isConnecting){
                connectTax(lastScan)
                sleep(5000)
```

```
                continue
            }
            if(taxiInstance.getForegroundState() && !isConnecting){
                btManagerDigitax?.DiscoveryStart(true, 5000)
                sleep(5000)
            }
            if (isConnecting){
                if (System.currentTimeMillis() - this.lastStatusReceived >
                    MAX_TIME_WITHOUT_STATUS){
                        isConnecting = false
                }
            }
        //enter data collection cycle
        } else {
            try {
                if (System.currentTimeMillis() - this.lastStatusReceived >
                    MAX_TIME_WITHOUT_STATUS){
                        isConnecting = false
                        this.txConnected = false
                        btDs = null
                        sleep(10000)
                        continue
                }
                taxManager!!.askTaximeterExtendedStatus(1)
                taxManager!!.askOdometerData(false)
                sleep(1000)
            } catch (exc: Exception) {
                sleep(5000)
            }
        }
    }
}
```

4.2.2 Hale Implementation

The implementation of the Hale brand taximeters required more involved work. While they have their own behavior documentation, Hale taximeters do not possess a pre-written API for application compatibility. As such, developing the communication and data collection procedures from the ground up is necessary.

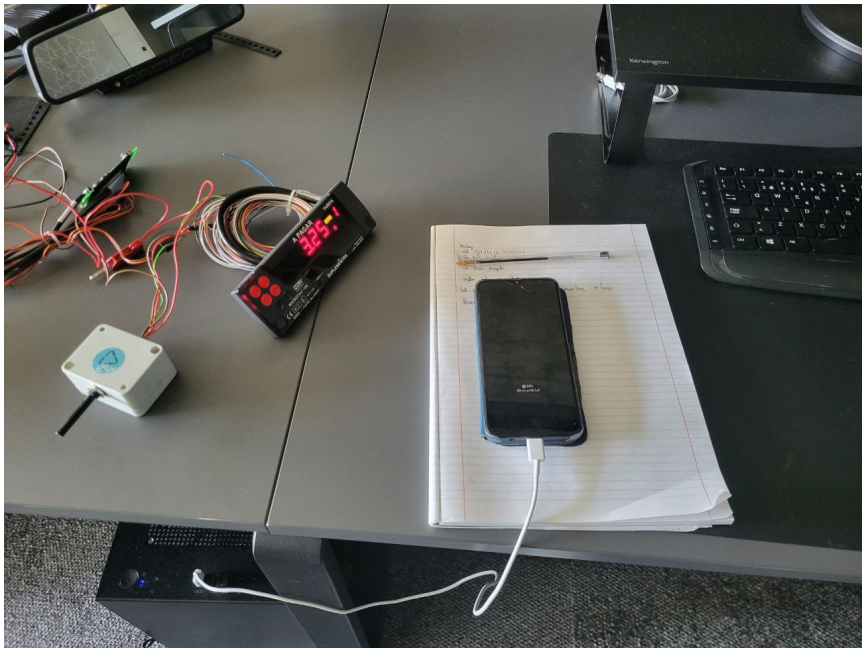


FIGURE 4.4: Hale taximeter and associated hardware

Hale taximeters, while very demanding on the software development side, were significantly easier to set up compared to the Digitax models due to coming with an integrated movement simulator. Figure 4.4 shows the final workspace for the brand. This workspace was far easier to set up due to not needing to program a Raspberry Pi and create a breadboard in order to simulate a trip, although it still required setting up the wiring properly according to the provided manual.

Hale taximeters' naming patterns depend on the model. In early development, a regular expression was used to search for all patterns, but once the server-side storage of which brand a driver uses began including the specific model for Hale brand taximeters, this search was streamlined only to include the correct naming pattern. The connection to these devices is then established using Kotlin's native Bluetooth API for paired devices.

Once the connection is established, Hale taximeters will use Bluetooth serial protocols to communicate with the application. The data sent by a Hale taximeter is limited by the license that the application possesses, and a license message must be sent to the taximeter

before it can send data regarding its current status, distance traveled, and on-screen information, such as its current fare. This data is sent in an encrypted format, meaning the application must decrypt it before it can be processed.

Hale taximeters have limitations regarding the data they send. Compared to Digitax taximeters, they lack the ability to send the current tariff type, meaning that certain fraud detection methods (using the wrong tariff in a given location, for example) can only be done through more complex computations.

Furthermore, from the driver's side, Hale taximeters face an issue that is not present with any other taximeter brand: their slow communication with the application. While every other taximeter brand's devices tend to respond nigh instantly, Hale devices can take up to 5 seconds to transmit any changes in their state or tariff through Bluetooth. While this may not seem like a very long time from an outside perspective, complaints regarding this issue were noted to be more frequent than any other project-related complaint.

```
override fun run(){
    while (true){
        sleep(1000)
        if (!txConnected && !isConnecting){
            startScan()
            sleep(1000)
            continue
        }
        if (isConnecting) {
            sleep(3000)
            continue
        }
        if(socket == null || socket?.isConnected != true) {
            txConnected = false
            emit_taximeter_connection_state_delayed()
            sleep(1000)
            continue
        }
        inputStream  = socket!!.inputStream
```

```
outputStream = socket!!.outputStream
if(!isReading)
    try {
        runBlocking {
            isReading = true
            launch(Dispatchers.IO){
                val data = readTMS3(inputStream!!)
                consumeInput(data)
                sleep(1000)
            }
        }
    } catch (e: Exception) {
        Timber.d("got exception $e")
        processSocketException(e)
    } finally {
        isReading = false
        sleep(1000)
    }
try {
    if (System.currentTimeMillis() - lastTotsRequest >
        MAX_TIME_WITHOUT_TOTS) {
            Timber.i("Requesting Totalizer data")
            outputStream!!.write(getTots())
            lastTotsRequest = System.currentTimeMillis()
    } else if ((currentTaxiState == 'O' ||
        (currentTaxiState == 'W' &&
        TaximeterRepository.tariffValue.value == 0.0f))
        && System.currentTimeMillis() - lastTripRequest >
        MAX_TIME_WITHOUT_TRIP) {
            Timber.i("Requesting Trip data")
            outputStream!!.write(getTrip())
            lastTripRequest = System.currentTimeMillis()
    } else if (currentTaxiState != 'O' &&
```

```
                System.currentTimeMillis() - lastTripRequest >
                MAX_TIME_WITHOUT_TRIP) {
                    Timber.i("Requesting Status data")
                    outputStream!!.write(getStatus())
                    lastTripRequest = System.currentTimeMillis()
            }
        }  catch (e: Exception) {
            Timber.d("got exception $e")
            processSocketException(e)
        } finally {
            sleep(1000)
        }
    }
}
```

4.2.3 Kienzle Implementation

The simplest taximeters by far, Kienzle brand taximeters, have minimal data collection capabilities. They use Bluetooth low-energy protocols for data transfer and only allow for the collection of whether they are occupied and the on-screen details of the current trip, if any.

Kienzle taximeters used a very similar hardware setup to the Digitax taximeters, shown in figure 4.3, but didn't require as much work due to already having the workspace prepared, and, as such, were significantly easier to integrate than the aforementioned brand.

Through Bluetooth low energy, data can be accessed by calling upon characteristics, each of which traditionally represents a different variable. Kienzle taximeters' data is all stored in a single characteristic, which changes its first character to signify what variable the characteristic refers to: the current state and, if applicable, tariff type, the current trip's fare, and the current trip's supplements.

Due to the aforementioned low data collection capabilities and small user count, Kienzle taximeters' entries were not saved to the server, as the data offered by their own protocols would not be usable for producing interesting results.

```
override fun run(){
    while (isRunning){
```

```
        if (System.currentTimeMillis() - lastNotificationReceived >
            TAXIMETER_NOTIFICATION_DISCONNECTED_DELAY &&
            lastNotificationReceived != 0L){
            txConnected = false
            emit_taximeter_connection_state()
        }
        if (!txConnected){
            scan_ble_devices()
            sleep(2000)
            continue
        }
        if (allCharacteristicLoaded){
            if (lastNotificationReceived == 0L ||
                System.currentTimeMillis() - lastNotificationReceived >
                BOX_UPDATE_TIMEOUT){
                readCharacteristic(txCharacteristic)
                sleep(TAXIMETER_SLEEP_DURATION.toLong())
            }
            verifyCharacteristicChanged()
        }
        sleep(250)
    }
}
```

4.2.4 Taxitronic Implementation

Taxitronic taximeters were the last brand to be implemented, and don't use Bluetooth to transmit their data to the mobile application directly. Instead, to obtain data from these devices, one must first launch and connect to them using a third-party application, SmartTD, which will then send intents that this project's mobile application can interpret.

Taxitronic taximeters were the easiest to work with from a hardware standpoint, as they only needed to be plugged into power to work and had a simple lever to simulate car movement.

FIGURE 4.5: Taxitronic taximeter as sent to Geolink

Intents are a way for mobile applications to communicate with each other and are broadcast to all other apps. To receive and interpret an intent, an action must be registered in an intent filter, which will launch an event if it receives one of its listed actions. An action is a short string that describes an intent; for example, a "TripComplete" action is sent from the SmartTD application and registered in the Taxi-Link Driver application's filter.

Taxitronic taximeters' intents transmit very limited data, being sent only when a trip is concluded and including only the trip's costs separated into tariff and supplement. Due to these limitations and not including a start or end date for the trip, these devices were not implemented with data-collecting capabilities beyond those needed for the driver's daily usage.

```
override fun run(){
    applicationContext.registerReceiver(eventListener, getIntentFilter())
    getOnChange()
    while (true){
        sleep(10000)
        getLastTrip()
    }
}
```

Chapter 5

Results

This chapter covers the results obtained in developing this project, which come in the form of Grafana dashboards that can be used to monitor taxis in real-time and in statistical data that was then analyzed.

5.1 Data Monitoring with Grafana

Monitoring data with Grafana allows for real-time data analysis; notable information can be and was obtained this way during this project's development. Some of the highlights of this information are:

- Digitax taximeters, despite their superior capabilities regarding data collection, can experience a glitch where their registered odometer distance (that is, the total distance traveled by the vehicle) resets to 0. This was noted to be unlikely to be intentional by both maker and user, as these resets almost always happen while a driver is in movement.

- Drivers' taxi odometers and tariffs were found with wrongful calibrations, with a specific case using millimeters instead of meters for both the odometer and value increases. This meant that while the values increased properly when using the calculations for €/Km of every other driver, this one appeared to be performing strangely poorly for the amount of money they were earning.

The dashboards developed with Grafana have several uses for taxi fleet managers and Geolink itself. The state collection, for example, allows for real-time monitoring of a given driver or group's progress. Figure 5.1 shows one of the dashboards developed for keeping

track of singular drivers, with thresholds for what constitutes "good" and "poor" values for each metric in the last hour. The €/Km gauge uses fixed thresholds, created with consultation from fleet managers, and the hourly tariff and supplement sums have a bar graph showing the comparison of previously registered hours of the same day.

FIGURE 5.1: A quick summary is given upon selecting a license and timespan (omitted for privacy).

Below the hourly and total summary given by the dashboard lies a list of every state instance recorded by the server, the distance traveled in total for each state, and the portion of time spent in it. This allows for observing any erratic behavior in taximeters, such as the lack of a payment state after being occupied in figure 5.2. The combination of these two sets of visualizations allows for both a quick summary of a taxi driver's performance along the chosen time interval and a more detailed analysis of their taximeter's behavior during shifts.

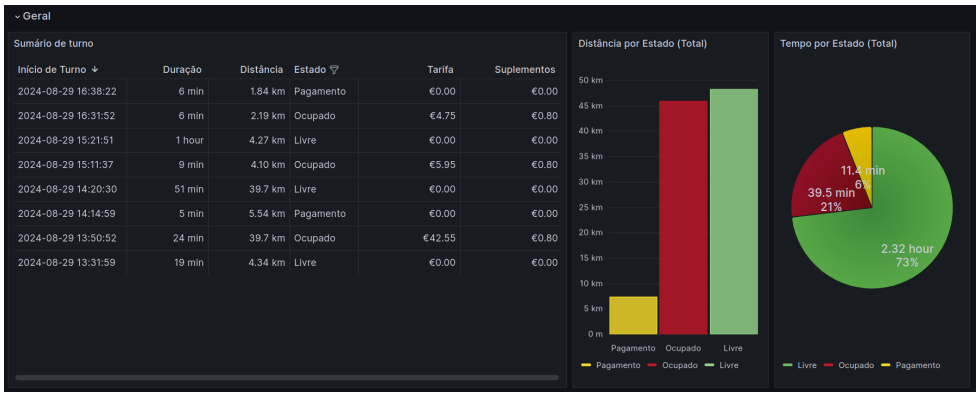


FIGURE 5.2: Below the summary lies a detailed list of the states registered to the collection.

Beyond the single driver dashboard, there are also two more dashboards, one of which is already being used professionally: the billing dashboard and the fleet dashboard.

The billing dashboard uses the trips dataset to show every trip a driver has made, indexed and also marks if a driver has billed that trip after completing it. If a trip is billed

later, a bill can either be emitted directly from the dashboard or annexed using services not developed for this project. As shown in figure 5.3, this dashboard is fully configured with Portuguese users in mind and has a very simplified UI for ease of use by technically non-savvy users.

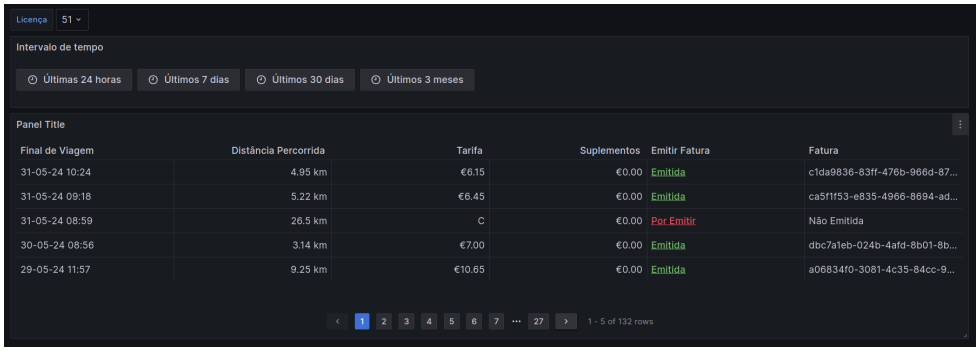


FIGURE 5.3: The billing dashboard allows for easy bookkeeping of the trips of each driver for their manager.

The last dashboard, which allows analysing whole fleets or even cities, allows for choosing a group of drivers and analysing all of their respective metrics, pictured in figure 5.4, shows the total earnings of a fleet, and even allows for filtering through the different types of supplements (called by application, required animal transport, etc), minimum distances, and allows for comparing each driver in the chosen group with their average tariff, supplements, and distance travelled per trip.

FIGURE 5.4: The fleet dashboard focuses on large scale analysis of taxi drivers

5.2 Statistical Analysis of Obtained Records

During the development of this project, a large volume of data was collected into two collections: state and trip. The state collection's data begins on December 11th, 2023, and the trip collection's data begins on January 1st, 2024; the last entry used for this thesis was obtained on August 14th, 2024.

This data was analyzed with the usage of Python's *pandas*, *numpy*, and *matplotlib* packages, along with complementary packages for easier visualization.

5.2.1 Initial Data Analysis

The first evaluation comes in the study of how long a driver tends to drive in each state (free, occupied, waiting for payment) and how long each state tends to last. This calculation was done over the base dataset.

The main statistics to be taken into account from the base dataset are as follows:

- The distance traveled during a trip, measured in kilometers;
- The time spent on a trip, measured in hours, minutes and seconds;
- The total fare earned during a trip, that is, the sum of the tariff and extras, in euros.

State	mD (km)	Dstd (km)	mT (m:s)	Tstd (h:m:s)	mF (€)	Fstd (€)
Free	2.10	504.67	21:24	4:34:16	NA	NA
Occupied	5.389	9.42	4:54	0:11:00	8.89	11.53
Waiting	5.389	9.42	1:50	0:13:36	NA	NA

TABLE 5.1: Unfiltered statistics from the dataset

From table 5.1, it is easy to see that there is a very high variance in every relevant field in the dataset. This is attributed to issues such as machine error, previous server iterations' lack of protection against mishandled data, and even misaligned taximeters. The usage of this raw data would most likely not only be accurate to the events occurring in reality but would even be counter-productive when used to guide taxi companies' analysis and strategies for their own work.

It is, however, important to note some of the base dataset's statistical information: notably, there are taxi drivers that spend very little time on average between trips and taxi drivers that can spend hours (not counting out-of-the-clock time) without doing a single service. According to some drivers' and call center workers' testimonies, this is

due to them not wanting to spend their time on low-paying trips when a more rewarding trip could appear the moment they leave.

5.2.2 Pre-processing

As explained in 5.2.1, there are multiple occurrences that can cause some of the data sent to the server to be faulty and erroneous. As such, there were several criteria used to filter out data entries from the state collection, which was generated using iterative code. The trip collection, having been accumulated using data sourced by the Digitax API and not running the risk of sending out-of-order data due to its in-built incremental IDs, did not require filtering, only treating the data values differently due to different units. The filtering done for the state analysis of paid trips was as follows:

- The minimum accepted distance was 200 meters, as this would take into account small packet losses;
- The euros per kilometer, explained later in this chapter, of each trip must be close to the legal parameters;
- The duration of each trip must be at least 1 minute;
- Test accounts' records are removed.

Beyond the filtering done for pre-processing, there is also a chance for obtaining more complex data by deriving information from pre-existing variables. The two most important derived characteristics for this study are as follows:

The city in which the taximeter operates can be obtained by cross-checking the taxi ID with Taxi-Link's own database. This can help study the differences between cities for other variables (if certain cities tend to have longer trips, for example), differences in adhesion to the usage of compatible taximeters, and differences in the number of trips a driver does on average.

The euros per kilometer (€/Km) earned during a trip allow for easier tracking of the gains and expenses of each taxi: each taxi will have a cost associated with the kilometers they travel, and the shorter a taxi trip is, or the more wait time there is without movement, the more efficient it is in euros per kilometer, as dictated by the taximeter fare conventions in Portugal[52].

The occupied statistics can be compared from the unfiltered dataset (when occupied) to the filtered dataset in order to see how much impact this filtering has on the recorded data:

Dataset	mD (km)	Dstd (km)	mT (m:s)	Tstd (m:s)	mF (€)	Fstd (€)	mEKm	EKmstd
Unfiltered	2.63	7.10	4:54	11:00	8.89	11.53	∞	NaN
Filtered	5.39	9.42	9:53	14:03	8.74	10.73	2.52	2.02

TABLE 5.2: Comparison between the filtered and unfiltered datasets

As seen in table 5.2, the filtering process has a noticeable impact on the means and deviations of every numerical field. However, while the increase in variance for both distance and time length of a trip might indicate that the dataset has grown more unstable, it has instead removed much more of the near-0 value entries than the higher value ones, which are, in fact, accurate to the real world taxi environment. This is most noticeable in the €/Km field, which experiences a shift from being unusable to being one of the most consistent fields in the dataset. This change can be best observed in figure 5.5, which highlights how much impact the filtering process has on this field.

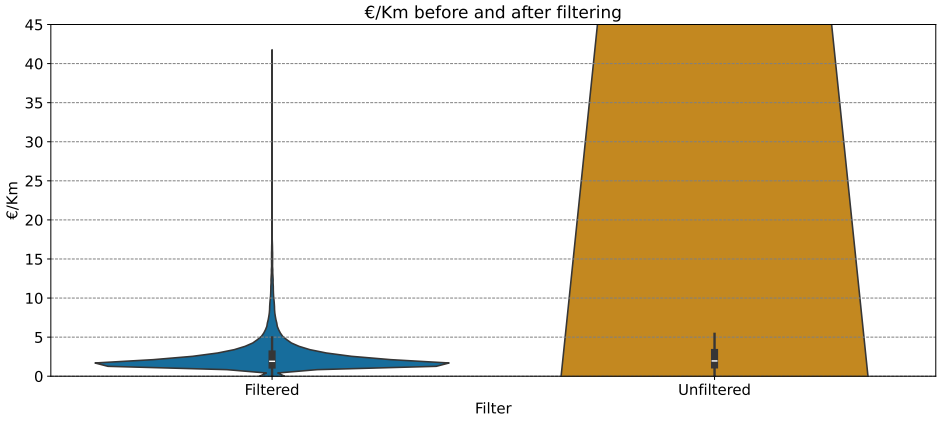


FIGURE 5.5: The impact of filtering on the €/Km statistic.

5.2.3 Global Analysis

As highlighted by figure 5.5, the euros per kilometer earned during a trip have a very high concentration below 5€/km, with only rare exceptions exceeding this value. This is, in part, affected by the fact that a taxi trip's value increases very slowly with time spent waiting[52], and has a lower ratio of euros per kilometer the farther the driver goes. Table 5.3 highlights this, and shows that even if a taxi trip's value increases with time, since

these longer trips most often don't include much waiting in place, longer-lasting trips end up being negatively correlated with the €/Km ratio.

Statistic	Distance	Length	Fare	€/Km
Distance	1	0.75	0.74	-0.25
Length	0.75	1	0.52	-0.22
Fare	0.74	0.52	1	0.04
€/Km	-0.25	-0.22	0.04	1

TABLE 5.3: Correlation table for the states dataset

All of the values obtained for table 5.3 were noted to be statistically significant with a null hypothesis test (p-value $<= 0.05$).

The trips module explained in subsection 3.3.2 compares a taximeter's recorded trips and the trips reported to the server. This comparison is shown in table 5.4, which places the trips API's recorded values against the states API's recorded values.

Dataset	**mD (km)**	**Dstd (km)**	**mT (m:s)**	**Tstd (h:m:s)**	**mF (€)**	**Fstd (€)**	**mEKm**	**EKmstd**
States API	5.39	9.42	9:53	0:14:03	8.74	10.73	2.52	2.02
Trips API	11.40	72.15	39:43	6:53:14	8.99	12.45	3.07	50.00

TABLE 5.4: States vs Trips API comparison

The results from the trips API display far greater standard deviations than the base dataset in every field *except* for the earned and far greater averages in both duration and distance traveled on average. This is attributed to drivers turning off their application for exceedingly long trips, as they feel no need to keep it open. However, the fare value being so consistent is attributed to many "street services" having low value and keeping the average from being too high, as seen in figure 5.6. The taxi performs these street services by being called by a pedestrian while at a taxi stop, as opposed to being called by Taxilink's server services. In later showcases of the trips API's data, it is safe to assume that the standard deviation for every numerical value is much higher than the states API's due to these long-duration trips.

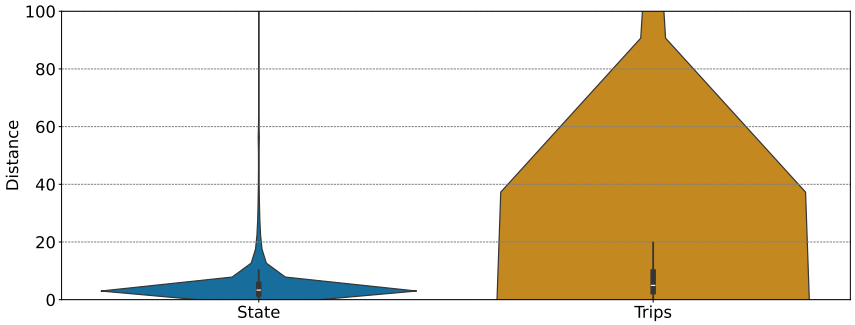
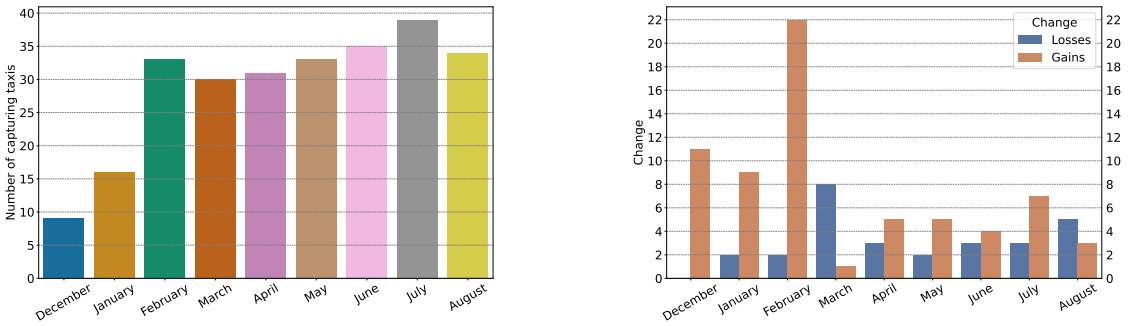


FIGURE 5.6: Distribution of trip distances traveled between the state and trip collections.

5.2.4 Driver Adherence

As the project developed, more taxi drivers began adhering to the usage of the capabilities provided by this project. This adherence is, as mentioned in chapter 2, stimulated by the decree-law passed on October 2023[12], which will come into effect on November 2024.

This adherence was not, however, immediate nor total: many more taxis still do not use this application's Bluetooth functionalities than those that do, and their adherence is still increasing as of the time of writing.



(A) Capturing taxis per month

(B) Monthly driver count changes

FIGURE 5.7: Number of taxis capturing data each month and how their number shifted.

In figure 5.7, the number of adhering taxis per month and the changes in these numbers can be observed. It is important to note that although February had large spikes in adherence, some of these new (to the project) taxi drivers had incompatible phones, as the functionalities provided require an Android version unavailable on older devices. As such, March appears to suffer a somewhat steep drop-off, as these adhering drivers needed to quit using the new functionalities provided.

Once these limitations were understood and potential adherents were warned, however, the increase in adherents stabilized, although some had to have their taximeter connection disabled or stopped working due to external reasons, such as Hale's general recall

of their MCT-06 model taximeters during July.

5.2.5 Monthly Analysis

In order to track how taxi activity changes each month, a categorized analysis can be performed, taking each trip's start time as the metric to determine which month it was in, even if it ended in a different month.

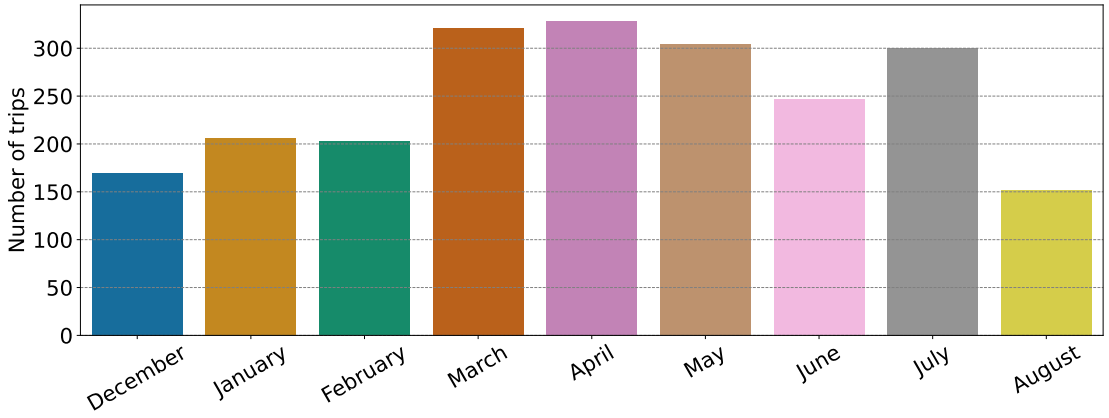


FIGURE 5.8: Distribution of average reported trips per driver per month.

The simplest analysis available is shown in figure 5.8, which shows how the number of trips per driver changes every month. From the aforementioned figure, it is easy to see that, regarding the capturing taxi drivers, August and December were the months with the least activity, and April and March were the months with the highest activity. However, it is important to note that December entries, post filtering, start on December 11th, 2023, and August entries end on August 14, 2024, as that is when the dataset was last updated before writing. By applying simple cross-multiplication, figure 5.9 is obtained, which shows how the average trip count would have progressed if the results had stayed consistent with the average for the missing days of each month.

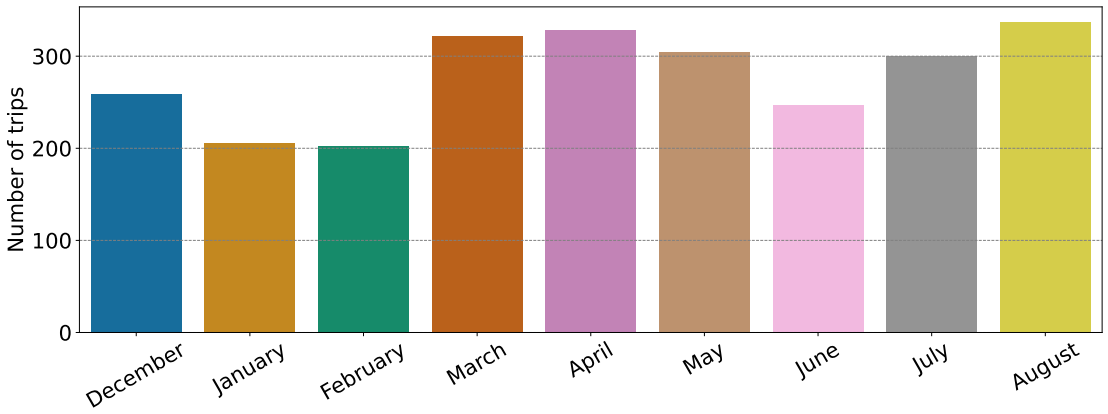


FIGURE 5.9: Cross-multiplied average reported trips per driver per month.

The change in how many trips were performed in December and August is stark, to the point where August now surpasses April (337 and 329 trips, respectively) in average trips per driver. When taking into account how many drivers each month has available, this would mean that either of these months would still have fewer total recorded trips than July. This, however, is a sign of a flaw in terms of retaining adherence from taxi drivers to the system, as the drivers who opt out of the functionalities provided by this system are still active.

Month	mD (km)	Dstd (km)	mT (m:s)	Tstd (m:s)	mF (€)	Fstd (€)	mEKm	EKmstd	AT/D
December	4.79	7.58	9:23	8:46	7.45	7.86	2.12	1.20	172
January	5.14	7.41	9:49	11.06	7.40	6.53	2.12	1.39	206
February	6.36	13.56	11:55	27.09	8.39	10.62	2.08	1.31	203
March	5.58	9.39	10:44	13:29	8.38	8.39	2.30	1.62	321
April	5.27	9.05	9:49	12:35	8.85	11.16	2.69	2.28	329
May	5.35	10.11	9:39	11:53	9.12	12.36	2.70	2.25	304
June	5.19	8.42	9:09	10:19	8.80	11.53	2.68	2.22	247
July	5.25	8.17	9:17	10:36	8.80	9.99	2.62	2.17	300
August	5.10	8.05	9:02	11:39	9.41	13.05	2.69	2.18	152

TABLE 5.5: Important metrics aggregated by month

The number of trips and adhering taxi drivers is not the only statistic that changes with the months. Table 5.5 shows considerable differences between the average travel distances and the months' travel times, even those that are consecutive. Furthermore, we can see that the winter months (December, January, and February) tend to have lower average earnings per kilometer than the rest of the observed months, with a difference of almost 10% to the closest month. These months also have 3 of the 4 lowest average fares in the dataset, despite February having the highest average distance in the whole dataset, with a 14% difference from March, which has the second highest. This is best

illustrated by figure 5.10, which further highlights how March, being a "middle ground" month between the winter months and the rest, is closer to winter than the rest in terms of its mean efficiency.

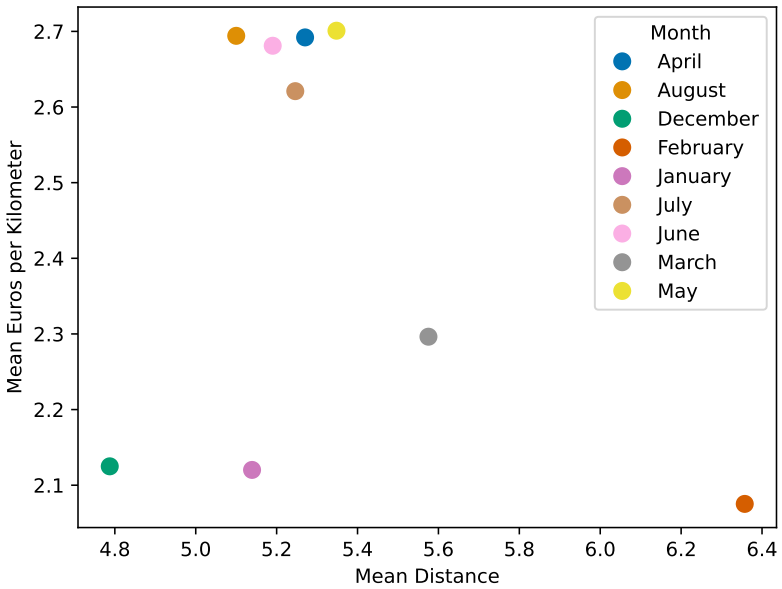


FIGURE 5.10: Distribution of mean trip distance and euros per kilometer of each month.

These fares per kilometer statistics also align with the fact that these months had, if one takes into account December and August's missing days, some of the lowest trip counts per driver, with February coming in last and January a very close second to last. December also has a very low trip count, although it would theoretically surpass June if the full month had been recorded.

Thanks to the trips module, it also becomes possible to compare the number of (accurately) reported trips to the number of real trips reported by the taximeter. The trips module was capable of accurately retrieving records starting in May, allowing for the comparison between the months of May, June, July and August seen in table 5.6. This comparison is limited to Digitax taximeters, as explained in subsection 3.3.2, but is still useful for establishing a baseline between the "real" values and the ones reported to the server.

Month	Trips mD (km)	State mD (km)	Trips mF (€)	State mF (€)	Trips AT/D	State AT/D
May	13.08	5.35	9.01	9.14	319	311
June	12.59	5.19	9.15	9.09	466	261
July	9.69	5.24	8.94	8.80	559	344
August	7.90	5.10	8.66	9.41	237	152

TABLE 5.6: Comparison between the trips and states APIs' datasets

The observed difference in the number of trips and the average distance traveled between the state and trip datasets highlights that, while the average value of the trips the drivers engage in is accurately reported, the distance they tend to travel is quite a bit higher than the reported one, which can be attributed either to an error in the API documentation, the server's recording system, or in the way the taximeter calculates a given trip's distance. Furthermore, the average number of trips for every month beyond May is always at least 50% higher than the one observed by the states API.

From this, we can tell that taxi drivers are under-reporting their trips by quite a lot, at least when it comes to digital means. As such, managers should be advised to pay special attention to how many trips a driver is reporting and comparing those to both their expected number and the numbers made available using platforms such as the one showcased in section 5.1.

5.2.6 City Analysis

A second useful split of the dataset lies in the analysis of how each city can behave and how many taxi drivers have adhered to the project from each recorded city. Several cities have adhering drivers, although the disparity between driver count between these cities is vast, as seen in figure 5.11. Furthermore, as shown in figure 5.7b, not all drivers who adhere to the project remain adherent for a multitude of reasons. Included in figure 5.11 is a comparison between the total number of taxis and the number of taxis each city had active during August. Notably, Lisbon was the city with the greatest disparity between the number of different drivers who had adhered and the number of adhering taxis active in August, with only 3 active drivers from the total 13 who tried the application.

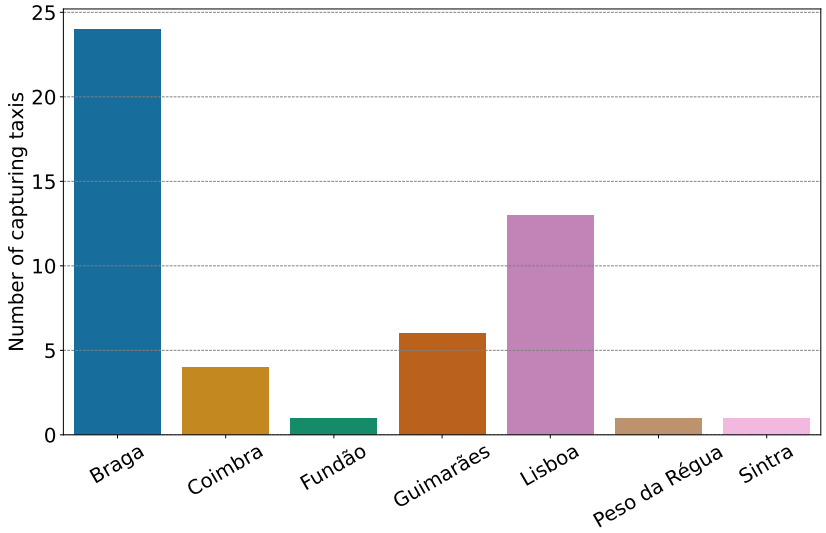


FIGURE 5.11: Total and final number of drivers per city

Due to the regular shifts in driver count, and different cities having different dates of joining, measuring the number of trips per city is not as valid a statistic as it is with months. However, comparing the statistics of trips themselves, that is, the distances, time lengths, fares, and €/Km is still quite valid, as these are not as affected by the drivers' adherence's stability.

City	mD (km)	Dstd (km)	mT (m:s)	Tstd (m:s)	mF (€)	Fstd (€)	mEKm	EKmstd
Braga	5.05	8.91	9:04	11:13	8.67	10.42	2.57	1.96
Coimbra	5.14	7.41	8:11	7:32	7.54	7.33	2.46	1.94
Fundão	6.36	13.56	10:34	10:37	11.74	10.43	2.80	2.49
Guimarães	5.58	9.39	15:05	33:45	9.65	11.64	2.30	2.32
Lisboa	5.27	9.05	9:49	15:59	16:29	10.63	2.13	2.15
Peso da Régua	5.35	10.11	13:22	20:09	11.38	21.34	2.66	2.69
Sintra	5.19	8.42	33:48	25:09	8.81	5.45	1.71	1.90

TABLE 5.7: Important metrics aggregated by city

Table 5.7 shows how each city compares to others regarding its most important metrics and highlights some interesting distinctions between each of them. Notably, while the breadth of the average distances is contained in the interval of 5.05 km to 6.36 km, there is a far greater variation in the standard deviation of the very same field, with Fundão approaching double the deviation of Coimbra, which is explained when looking at the distribution of distances shown in figure 5.12.

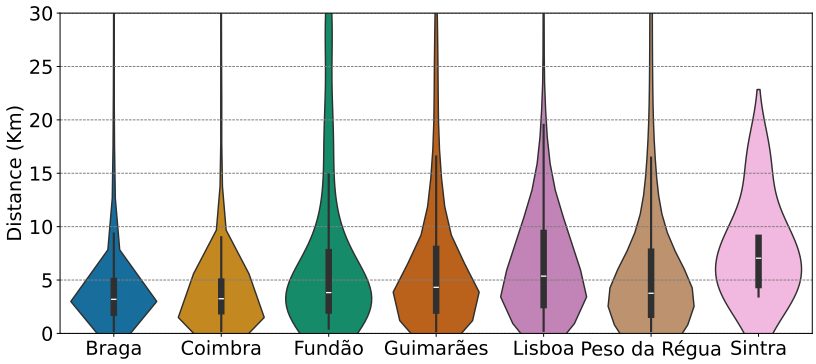


FIGURE 5.12: Violin plot of the distances traveled for each city

Despite this low difference in mean travel distances, however, there is a notable difference in both the mean €/Km and, thus, earned fares for each city, with Fundão displaying a 63.7% increase compared to Sintra. These differences are attributed to a city's trips' mean duration, as the lower this value is, the higher the mean €/Km tends to be, as highlighted in figure 5.13 and noted in table 5.3.

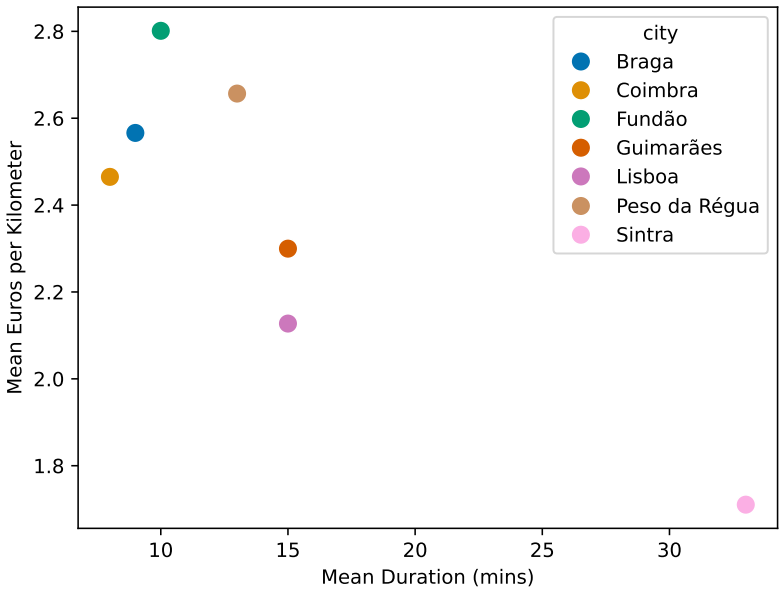


FIGURE 5.13: Relation between €/Km and trip duration.

As with the analysis of monthly data, the comparison between the state and trips dataset can be used to highlight any incongruities between the trips that were recorded by the taximeter and those that were reported to the server. Sintra is not included in this analysis as its reporting driver did not have a Digitax taximeter installed.

City	Trips mD (km)	State mD (km)	Trips mF (€)	State mF (€)	Trips m€/Km	State m€/Km
Braga	7.74	5.04	8.82	9.01	2.94	2.71
Coimbra	6.97	4.50	7.73	7.70	1.66	2.54
Fundão	31.67	6.86	17.65	11.74	1.18	2.80
Guimarães	14.03	6.79	9.69	9.83	3.86	2.48
Lisboa	37.52	6.80	9.79	9.86	3.60	2.45
Peso da Régua	13.83	7.27	12.13	12.67	8.69	2.86

TABLE 5.8: Comparison between the trips and states APIs' datasets

The discrepancies between the trips and state datasets are notable in all fields except for Braga and Coimbra, which instead display (comparatively) very small differences in their values. When one takes into account the increase in **percentages** of the base value, three groups are immediately formed: Braga and Coimbra, which both display roughly 50% increases, Guimarães and Peso da Régua, which display nearly 100% increases, and Fundão and Lisboa, which display over 300% increases. Despite this tremendous increase in mean distances, only Fundão displays a significant change in its mean fare, whereas every other city sees a change of less than 1€. On the other hand, despite these differences in mean distance with a similar mean fare, the average €/Km remains relatively close for Lisbon. Peso da Régua, on the other hand, experiences a very dissimilar phenomenon. Its average distance nearly doubles, and its average fare is, in fact, *lower* than the reported one, but its average €/Km is more than triple its reported value, which is similar to Braga's case, albeit on a much more easily observed scale. This is all caused by the very high volatility of trips mentioned in subsection 5.2.3, with rare trips with seemingly normal values for both fields having a proportion that heavily distorts the €/Km statistics.

Despite these outlying trips, it is important to remember that the trips dataset is collected directly from the taximeter's internal storage through Digitax's proprietary API. As such, it is safe to assume that these disproportional trips are both legal and accurately reported, outstanding as their numbers might be.

Chapter 6

Conclusion

The taxi industry needs to adopt new methods and technologies if it hopes to keep up with the rivals the new digital age has wrought in ride-hailing applications, and studies agree that adopting digital platforms to complement their services will help them in this endeavor. There are studies being compiled every day researching the best option for this challenge, both from a competitive standpoint by making the service preferable to its ride-hailing counterparts and from a legal standpoint to limit the infrastructural and environmental impact wrought by these digital services.

In this work, we have designed an adaptable infrastructure that can facilitate competition and ensure that taxis can continue operating as usual despite new regulations. This infrastructure, consisting of a mobile app that connects to taximeters via Bluetooth and a web server for data storage and visualization, can be expanded to incorporate additional features as the industry evolves.

Moreover, we have conducted a statistical analysis of the data collected by this infrastructure and presented the visual aids available to those using the services developed during this work. These aids, already in use by a taxi fleet, demonstrate the practicality and effectiveness of the proposed infrastructure, instilling confidence in its potential to enhance the industry.

The biggest challenge faced was taxi drivers not adhering to this project, as many prefer not to report many of their trips or even adhere to the project at all, and understandably so, given the costs of installing Bluetooth-compatible taximeters. However, this is a challenge that only time can overcome, as November will herald the enforcement of the law established in October 2023 and a theorized higher adherence rate.

6.1 Future Work

There are many potential uses for the infrastructure developed for this work, both from an academic perspective by collecting data for analysis and from a professional perspective to aid the taxi industry. The most immediate goal is to integrate the new infrastructure fully with Geolink's pre-existing infrastructure, such as linking the monitoring software so the new driver data is stored in conjunction with the non-taximeter related data that was already being stored, such as car model to measure fuel consumption and other useful details.

Another potential avenue for continuing these studies is to create an even more streamlined MongoDB collection for the data sent by all taximeter brands, taking into account the limitations of all taximeter brands to store the possible relevant details.

Bibliography

[1] C. Miguel, E. Martos-Carrión, and M. Santa, *A Conceptualisation of the Sharing Economy: Towards Theoretical Meaningfulness*. Cham: Springer International Publishing, 2022, pp. 21–40. [Online]. Available: https://doi.org/10.1007/978-3-030-86897-0_2 [Cited on page 1.]

[2] G. Willis and E. Tranos, "Using 'big data' to understand the impacts of uber on taxis in new york city," *Travel Behaviour and Society*, vol. 22, pp. 94–107, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214367X20302027 [Cited on pages vi and 1.]

[3] Álvaro Aguilera-García, J. Gomez, G. Velázquez, and J. M. Vassallo, "Ridesourcing vs. traditional taxi services: Understanding users' choices and preferences in spain," *Transportation Research Part A: Policy and Practice*, vol. 155, pp. 161–178, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0965856421002809 [Cited on page 1.]

[4] S. Guo, Y. Liu, K. Xu, and D. M. Chiu, "Understanding passenger reaction to dynamic prices in ride-on-demand service," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2017, pp. 42–45. [Cited on page 1.]

[5] S. Jiang, L. Chen, A. Mislove, and C. Wilson, "On ridesharing competition and accessibility: Evidence from uber, lyft, and taxi," in *Proceedings of the 2018 World Wide Web Conference*, ser. WWW '18. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018, p. 863–872. [Online]. Available: https://doi.org/10.1145/3178876.3186134 [Cited on page 1.]

[6] T. Berger, C. Chen, and C. B. Frey, "Drivers of disruption? estimating the uber effect," *European Economic Review*, vol. 110, pp. 197–210, 2018. [Online]. Available:

https://www.sciencedirect.com/science/article/pii/S0014292118300849 [Cited on pages 2 and 4.]

[7] M. Tarduno, "The congestion costs of uber and lyft," *Journal of Urban Economics*, vol. 122, p. 103318, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094119020300899 [Cited on page 2.]

[8] D. Anair, J. Martin, M. C. P. de Moura, and J. Goldman, "Ride-hailing's climate risks: Steering a growing industry toward a clean transportation future," *Union of Concerned Scientists*, 2020. [Online]. Available: https://www.ucsusa.org/resources/ride-hailing-climate-risks [Cited on page 2.]

[9] R. B. Collier, V. Dubal, and C. L. Carter, "Disrupting regulation, regulating disruption: The politics of uber in the united states," *Perspectives on Politics*, vol. 16, no. 4, p. 919–937, 2018. [Cited on page 2.]

[10] P. Pelzer, K. Frenken, and W. Boon, "Institutional entrepreneurship in the platform economy: How uber tried (and failed) to change the dutch taxi law," *Environmental Innovation and Societal Transitions*, vol. 33, pp. 1–12, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210422418301631 [Cited on page 2.]

[11] R. Calo and A. Rosenblat, "The taking economy: Uber, information, and power," *Columbia Law Review*, vol. 117, 2017, university of Washington School of Law Research Paper No. 2017-08. [Online]. Available: https://ssrn.com/abstract=2929643 [Cited on page 2.]

[12] D. da República, "Decreto-lei n.º 101/2023," https://diariodarepublica.pt/dr/detalhe/decreto-lei/101-2023-223575032, 2023, accessed: 2024-08-26. [Cited on pages 2 and 41.]

[13] J. Gomez, Álvaro Aguilera-García, F. F. Dias, C. R. Bhat, and J. M. Vassallo, "Adoption and frequency of use of ride-hailing services in a european city: The case of madrid," *Transportation Research Part C: Emerging Technologies*, vol. 131, p. 103359, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X21003612 [Cited on page 4.]

[14] L. Pepić, "The sharing economy: Uber and its effect on taxi companies," *ACTA ECONOMICA*, vol. 16, 12 2018. [Cited on page 4.]

[15] J. Cramer and A. Krueger, "Disruptive change in the taxi business: The case of uber †," *American Economic Review*, vol. 106, pp. 177–182, 05 2016. [Cited on page 4.]

[16] G. R. Fréchette, A. Lizzeri, and T. Salz, "Frictions in a competitive, regulated market: Evidence from taxis," *American Economic Review*, vol. 109, no. 8, p. 2954–92, August 2019. [Online]. Available: https://www.aeaweb.org/articles?id=10.1257/aer.20161720 [Cited on page 4.]

[17] G. Dudley, D. Banister, and T. Schwanen, "The rise of uber and regulating the disruptive innovator," *The Political Quarterly*, vol. 88, 05 2017. [Cited on page 4.]

[18] K. Väyrynen and A. Lanamäki, "Policy ambiguity and regulative legitimacy of technology: Legal indeterminacy as result of an ambiguous taximeter regulation," in *International Conference on Information Systems 2020*, 12 2020. [Cited on page 4.]

[19] S. Harding, M. Kandlikar, and S. Gulati, "Taxi apps, regulation, and the market for taxi journeys," *Transportation Research Part A: Policy and Practice*, vol. 88, pp. 15–25, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0965856416302191 [Cited on pages 5 and 6.]

[20] J. Cooper and R. Mundy, *Taxi! Urban economies and the social and transport impacts of the taxicab*. Routledge, 2016. [Cited on page 5.]

[21] K. Thelen, "Regulating uber: The politics of the platform economy in europe and the united states," *Perspectives on Politics*, vol. 16, no. 4, pp. 938–953, 2018. [Cited on page 5.]

[22] P. F. Martins, "Sharing economy, competition and regulation: the case of uber in the case-law of the court of justice of the european union," pp. 54–67, 2019. [Cited on page 5.]

[23] T. Gu, W. Xu, P. Shi, R. Wang, and I. Kim, "Taxi in competition with online car-hailing drivers: Policy implication to operating strategies," *Multimodal Transportation*, vol. 3, no. 2, p. 100129, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2772586324000108 [Cited on page 5.]

[24] S. Guo, Y. Liu, K. Xu, and D. M. Chiu, "Understanding ride-on-demand service: Demand and dynamic pricing," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2017, pp. 509–514. [Cited on page 5.]

[25] C. Yan, H. Zhu, N. Korolko, and D. Woodard, "Dynamic pricing and matching in ride-hailing platforms," *Naval Research Logistics (NRL)*, vol. 67, 11 2019. [Cited on page 5.]

[26] X. Qian and S. Ukkusuri, "Time-of-day pricing in taxi markets," *IEEE Transactions on Intelligent Transportation Systems*, vol. PP, 01 2017. [Cited on page 5.]

[27] K. Xu, M. Saberi, and W. Liu, "Dynamic pricing and penalty strategies in a coupled market with ridesourcing service and taxi considering time-dependent order cancellation behaviour," *Transportation Research Part C: Emerging Technologies*, vol. 138, p. 103621, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X22000663 [Cited on page 5.]

[28] Y.-M. Jin, X.-F. Ye, W.-L. Liu, T. Wang, and H. Wang, "Dynamic pricing model for cruising taxicab based on system dynamics," *Advances in Mechanical Engineering*, vol. 11, p. 168781401982711, 02 2019. [Cited on pages vi, 5, and 6.]

[29] M. Egan, J. Drchal, J. Mrkos, and M. Jakob, "Towards data-driven on-demand transport," *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, vol. 5, p. 154835, 06 2018. [Cited on page 5.]

[30] M. Maljkovic, G. Nilsson, and N. Geroliminis, "Hierarchical pricing game for balancing the charging of ride-hailing electric fleets," *IEEE Transactions on Control Systems Technology*, vol. 31, no. 6, pp. 2728–2743, 2023. [Cited on page 6.]

[31] E. Tervo and K. Väyrynen, "Non-adoption of dynamic pricing in traditional taxi dispatch organizations," in *Proceedings of the 55th Hawaii International Conference on System Sciences*, 01 2022. [Cited on page 6.]

[32] S. Guo, C. Chen, J. Wang, Y. Liu, K. Xu, and D. M. Chiu, "Fine-grained dynamic price prediction in ride-on-demand services: Models and evaluations," *Mobile Networks and Applications*, vol. 25, 04 2020. [Cited on page 6.]

[33] D. Pandya and G. Kumar, "Applying industry 4.0 technologies for the sustainability of small service enterprises," *Service Business*, vol. 17, pp. 37–59, 3 2023. [Cited on page 6.]

[34] E. Calderon-Monge and D. Ribeiro-Soriano, "The role of digitalization in business and management: a systematic literature review," *Review of Managerial Science*, vol. 18, pp. 449–491, 2 2024. [Cited on page 6.]

[35] Y.-H. Shr and H.-H. Chang, "The effects of participating in digital ride-hailing on taxi drivers' business operations," *Transportation Research Part A: Policy and Practice*, vol. 187, p. 104167, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0965856424002155 [Cited on page 6.]

[36] N. Akbulaev, "The impact of the taxi service mobile applications on the financial condition of taxi companies," *International Journal of Scientific & Technology Research*, vol. 9, pp. 2144–2150, 02 2020. [Cited on page 7.]

[37] A. Brodeur and K. Nield, "An empirical analysis of taxi, lyft and uber rides: Evidence from weather shocks in nyc," *Journal of Economic Behavior & Organization*, vol. 152, pp. 1–16, 08 2018. [Cited on page 7.]

[38] D. Zhang, L. Sun, B. Li, C. Chen, G. Pan, S. Li, and Z. Wu, "Understanding taxi service strategies from taxi gps traces," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 1, pp. 123–135, 2015. [Cited on page 7.]

[39] R. Ozeki, H. Yonekura, H. Rizk, and H. Yamaguchi, "Balancing privacy and utility of spatio-temporal data for taxi-demand prediction," in *2023 24th IEEE International Conference on Mobile Data Management (MDM)*, 2023, pp. 215–220. [Cited on page 7.]

[40] A. Mae, M. Adriano, C. Co, and S. Sy, "Out with the old, in with the new: A study on the vehicle hailing preferences of filipino taxi riders based on participation intent," 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:218625805 [Cited on page 7.]

[41] J. DiMarzio, *Beginning android programming with android studio*. John Wiley & Sons, 2016. [Cited on page 7.]

[42] M. Wolfson and D. Felker, *Android developer tools essentials: Android Studio to Zipalign*. " O'Reilly Media, Inc.", 2013. [Cited on page 7.]

[43] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, 2014. [Cited on page 8.]

[44] E. Hahn, *Express in Action: Writing, Building, and Testing Node.Js Applications*. Shelter Island, NY: Manning Publications, 2016. [Cited on page 10.]

[45] M. Indrawan-Santiago, "Database research: Are we at a crossroad? reflection on nosql," in *2012 15th International Conference on Network-Based Information Systems*, 2012, pp. 45–51. [Cited on page 11.]

[46] R. Hecht and S. Jablonski, "Nosql evaluation: A use case oriented survey," in *2011 International Conference on Cloud and Service Computing*, 2011, pp. 336–341. [Cited on page 11.]

[47] C. A. Győrödi, D. V. Dumşe-Burescu, D. R. Zmaranda, and R. Győrödi, "A comparative study of mongodb and document-based mysql for big data application data management," *Big Data and Cognitive Computing*, vol. 6, 6 2022. [Cited on page 12.]

[48] A. Makris, K. Tserpes, G. Spiliopoulos, and D. Anagnostopoulos, "Performance evaluation of mongodb and postgresql for spatio-temporal data," 2019. [Cited on page 12.]

[49] A. Makris, K. Tserpes, G. Spiliopoulos, D. Zissis, and D. Anagnostopoulos, "Mongodb vs postgresql: A comparative study on performance aspects," *GeoInformatica*, vol. 25, pp. 243–268, 4 2021. [Cited on page 12.]

[50] S. Agarwal and K. S. Rajan, "Performance analysis of mongodb versus postgis/postgresql databases for line intersection and point containment spatial queries," *Spatial Information Research*, vol. 24, pp. 671–677, 12 2016. [Cited on page 12.]

[51] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995. [Cited on page 19.]

[52] P. Government, "Tarifários de táxis," https://eportugal.gov.pt/servicos/consultar-o-tarifario-de-taxis, accessed: 2024-08-21. [Cited on pages 38 and 39.]