XTraN Green Driving App

João Eduardo Alves Nogueira
joao.alves.nogueira@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2022

Abstract

In a world currently facing an energy crisis and still trying to fight off global warming, the levels of pollution caused by companies' vehicle fleets have a considerable negative impact on this problem. One way of mitigating this is for people to adopt environmentally friendly driving practices. This document describes the work developed as part of the *DriveGreen* project, proposed by the Portuguese company *Tecmic - Tecnologias de Microelectrónica, S.A.*.

The goal was the creation and integration of a new eco-driving support module in the company's *TeamMobile* mobile application. This application is used by company fleets' drivers belonging to Tecmic's clients, and provides features relating to GPS tracking and management of their overall daily work assignments. The new module receives the vehicle's sensor data from Tecmic's *XTraN* unit installed on it, and analyses and transforms that data into useful metrics related to ongoing and finished journeys: as a small set of charts in a bottom bar available during the trip, and through a detailed screen with suggestions on how to improve the user's driving style based on the finished journey data, respectively.

The module was developed and integrated in the *TeamMobile* application, but it is not yet finished, requiring additional tests related to its communication protocol with the *XTraN* unit and a new developed web service. After this, the new module will be subjected to internal usability tests, before being made available in a beta version to some of the company's clients. This will ultimately lead to the official release of this new version of the application.

Keywords: Mobile Application, Xamarin, Android, Eco-Driving.

1. Introduction

The *DriveGreen* project, proposed by the Portuguese company *Tecmic S.A.*, is a mobile application module whose goal is to retrieve and analyse vehicle sensor data and present it in the drivers' screen in the form of charts, statistics and driving suggestions, available during and after each journey. This is meant to help the user to improve his eco-driving style and also reduce fuel consumption. The module's communication protocol is based on direct communication with a *XTraN* unit placed on the vehicle, or with a web service that retrieves data sent from this unit to the company's central database. In the application the user has access to this data during the journey, as a set of four small charts placed in the background of the mobile application screens, and after the journey, with a new screen that shows the journey's summary data along with eco-driving suggestions.

Tecmic's *TeamMobile*, the system in which the new *DriveGreen* module was incorporated, is a mobile application used by the company clients' drivers. These drivers perform different services in their daily work, depending on the client company's field of operation, that require the driver and his team to travel around the country. These workers start their day by logging in the Activity screen and selecting the first journey of the day in the Journey screen, which also features a GPS navigation module. Each journey can contain multiple stops: different places where there is work to be done, and the application is used to register that work in different forms specific to each client. At the end of the day the user stops the Activity in its corresponding screen. We can see that this application is of massive importance to manage the workers' every day aspect, and given the long journeys that are executed, it is missing a eco-driving module, which this project aims to fix.

In this document we start by taking a look at related eco-driving systems as well as a briefly reviewing what eco-driving really is. We then present the system's architecture and implementation details, the module's testing and evaluation and finish with some conclusions about the whole process.

2. Related Work

Here we take a look at eco-driving impact and good practices, we present Tecmic's *Ecodriver* module for a different system, and we analyse and compare existing eco-driving mobile applications.

2.1. Eco-Driving

Before looking at some eco-driving support systems it is important to know what eco-driving really is. Among the multiple available meanings, one of the broader ones defines it as the "adoption of practices and behaviours that promote more energy-efficient, safe and environmentally-friendly driving" [1]. One way of contributing to this is to improve the driver's driving style, which is what this project aims to achieve.

The main aspects to ensure the practice of a eco-driving friendly style have to do with driving speed, acceleration, deceleration and idling. Concerning speed, the driver should try to maintain constant speed whenever possible and of course respect speed limits. As for acceleration and deceleration the driver should avoid aggressive braking and accelerating by practicing a cautious driving style, maintaining enough distance from other vehicles and trying to predict upcoming problems in the road. Finally, idling is a awful behavior which heavily contributes to fuel consumption and emission of air pollutants, and should be avoided by turning of the vehicle's engine whenever stuck in traffic or waiting for stoplights [2].

And do these methods actually yield results? According to several studies the answer is positive, for instance in Finland [3] where an eco-driving application used by Helsinki's bus drivers resulted in a 8.9% fuel consumption reduction as well as an increase in the passengers reported comfort. In Switzerland [4], fifty company fleet drivers also used a mobile application that provided feedback based on speed, acceleration, deceleration and gear usage, resulting in a 3.23% fuel consumption reduction.

2.2. Tecmic's *Ecodriver*

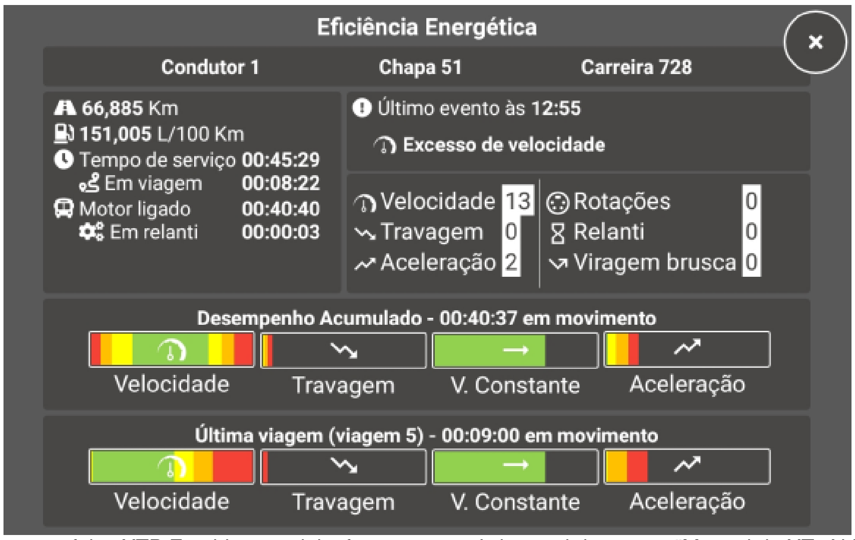
Tecmic's *XTraN Passenger* [5] is a passenger transport management system for which was developed a eco-driving support module, the *XTraN Passenger Ecodriver*. The module gathers information directly from the vehicle's *CAN-Bus* system, and after analysing and transforming the data presents it in the driver's on-board Android console. This information is also saved in the system's central database, which allows the creation of eco-driving reports available in the system's web platform.

The module's application layout includes a set of charts on a bottom bar visible in the main application screens, and a detailed screen with journey summary data. This second and main screen can be seen in figure 1.

This module's design and goals are very much similar to the current project's ones and as so *DriveGreen*'s layout and functionality is heavily based on it. However, one of the main features, not present in this system that this project aims to achieve, is giving its user feedback about how to improve his driving style, based on all the data that was gathered during each journey.

2.3. Eco-Driving Mobile Applications

Since other companies eco-driving systems details such as *Samsara*'s [6] or *INGTECH*'s [7] are made private, we decided to study some mobile Android or iOS applications regarding the same subject. This analysis was done based on a overview of the applications' functionality and layout design in comparison to the current project's goals.


Eficiência Energética

Condutor 1 Chapa 51 Carreira 728

66,885 Km
151,005 L/100 Km
Tempo de serviço 00:45:29
Em viagem 00:08:22
Motor ligado 00:40:40
Em relanti 00:00:03

Último evento às 12:55
Excesso de velocidade

Velocidade	13	Rotações	0
Travagem	0	Relanti	0
Aceleração	2	Viragem brusca	0

Desempenho Acumulado - 00:40:37 em movimento

Velocidade Travagem V. Constante Aceleração

Última viagem (viagem 5) - 00:09:00 em movimento

Velocidade Travagem V. Constante Aceleração

Figure 1: Main screen of the *XTP Ecodriver* module, from company's internal document "Manual do XTraN Passenger Web".

	Real Time Data	Persistent Data	Vehicle Sensor Data	Simple/Non-Intrusive Layout	Eco-Driving Data
Car Scanner	x		x		
Speedometer	x			x	x
Drive Eco	x	x			x
Ecologic	x	x	x		x
Torque Pro	x		x		
Dash Command	x		x		x
DriveGreen	x	x	x	x	x

Table 1: Comparison between the analysed applications' features and this project's goals.

Six applications were analysed: 1) Car Scanner ELM OBD2 [8], 2) ECO-Driving Speedometer [9], 3) Drive Eco [10], 4) Ecologic [11], 5) Torque Pro [12], and 6) DashCommand [13].

In table 1 we can see whether or not these applications meet all of the *DriveGreen* project's objectives.

We concluded that none of them met all of the project's goals. Most of the applications focus on collecting vehicle data using *OBD* (On-Board Diagnostics) and show a ton of complex data to the user, that he will simply not access or understand, much less during an ongoing journey. Only some of them contain features that allow the user to access the journey's summary data after it is finished, and even though all of them claim to provide eco-driving support characteristics, they are mostly pretty lackluster.

An application such as the *ECO-Driving Speedometer* [9], although extremely simple, meets the goal of a clean, easy to understand and non intrusive system that provides some eco-driving data. While an application such as *Ecologic* [11], even though presenting much more detailed data, including past journeys history, it is not adequate given its complexity and amount of data that the user will never really make use of.

We want *DriveGreen* to gather the vehicle's real-time data and present it to the user in a simple way that helps him improve his driving style, during and after his journeys.

3. System Features & Development

In this chapter we present several aspects regarding the architecture, features and overall development of this system.

3.1. Technological Stack & Xamarin

The system relied on the following main technologies for its development: a) Xamarin to create and add the new module to the existing *TeamMobile* application; b) The *MVVMCross* library also present in the main application; c) ASP.NET CORE for the development of the web service; d) Google's Protocol Buffers to serialize the data passed between the application and the *XTraN* unit; e) SQLite for the application's internal database.

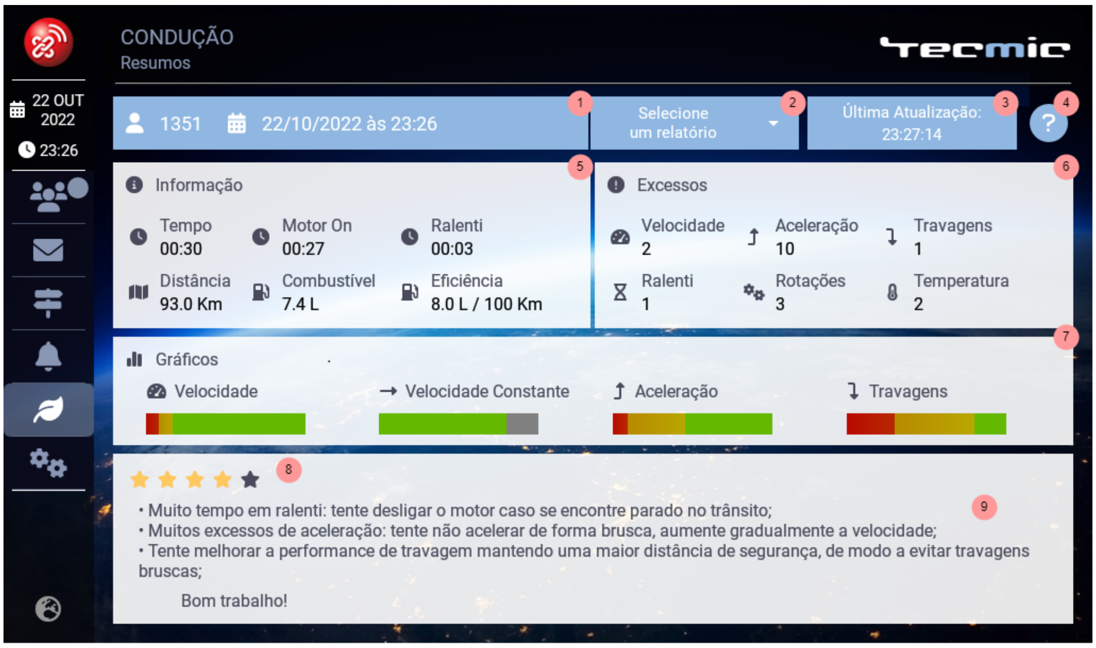
TeamMobile was developed using Native Xamarin, which is the reason why it is the main technology used in this project. Xamarin allows faster development of cross-platform applications, since all back-end code is shared, and only the screen's layout code has to be written separately for each desired platform [14]. On top of this, *TeamMobile* also makes use of the *MVVMCross* library [15], which features data-binding, dependency injection and unit testing tools, besides allowing the usage of the MVVM architectural pattern in the application's development. The MVVM pattern, standing for Model-View-ViewModel, separates the application's responsibilities between three main components, the Model which contains the business layer logic, the View which is the interface layout where data is shown, and the View-Model that connects these two components, defining the applications' logic. [16].

3.2. Eco-Driving Data

The vehicle's sensor data is obtained by the equipped *XTraN* unit, and further transformed by its *ECSM* (Energy, Comfort and Security Module) module into relevant statistics. Among the long list of provided measurements, the system will be using: a) Counters such as total fuel consumption, traveled distance, journey time, online engine time and idling time; b) Number of penalties during the journey, such as excessive speed, acceleration, deceleration, idling, engine temperature and engine rotations; c) Total journey time spent practicing speed, acceleration, deceleration, and constant speed behaviours included in predefined value ranges.

These last data values are used to create four charts corresponding to those same behaviours. These charts measure the drivers performance in each behaviour in a certain time window, meaning the last five minutes of the journey when displayed in the module's bottom bar, or the whole journey when shown in the module's main screen.

For the speed, acceleration and deceleration charts, we represent three different values, corresponding to adequate, acceptable and inadequate


CONDUÇÃO
Resumos

22 OUT 2022
23:26

1351 22/10/2022 às 23:26 | Selecione um relatório | Última Atualização: 23:27:14

Informação
Tempo 00:30
Motor On 00:27
Ralenti 00:03
Distância 93.0 Km
Combustível 7.4 L
Eficiência 8.0 L / 100 Km

Excessos
Velocidade 2
Aceleração 10
Travagens 1
Ralenti 1
Rotações 3
Temperatura 2

Gráficos
Velocidade
Velocidade Constante
Aceleração
Travagens

• Muito tempo em ralenti: tente desligar o motor caso se encontre parado no trânsito;
• Muitos excessos de aceleração: tente não acelerar de forma brusca, aumente gradualmente a velocidade;
• Tente melhorar a performance de travagem mantendo uma maior distância de segurança, de modo a evitar travagens bruscas;

Bom trabalho!

Figure 2: *DriveGreen*'s main screen.

behaviours. Their relative amounts are displayed according to the lengths of the green, yellow and red bars, respectively.

As for the constant speed chart there are only two different values, corresponding to whether or not the driver is at constant speed, once again represented by the lengths of green and grey bars.

Besides these, the module is also responsible for generating two new types of data, the journey's score and eco-driving suggestions. The score is calculated given the number of penalties committed on that journey, based on a threshold of twenty penalties for each 100 kilometers. The suggestions algorithm generates a maximum of three recommendations based on the journey's data. First it checks whether or not the total idling time surpasses 5% of the total journey time, and if so displays a first suggestion regarding this behaviour. Secondly it analyses the total journey penalties and displays a suggestion regarding the behaviour with the largest number of penalties. Lastly, the four charts are analysed and similarly to the last step, a suggestion is displayed for the behaviour in which the red bar size surpasses a certain threshold.

This basic algorithm was created keeping in mind some rules. We don't want the user to feel discouraged by the system, as such we display a maximum of three and minimum of zero recommendations, and use language such as "try to..." instead of straight orders, while also greeting the user after presenting these. Moreover, the generated recommendations are related with the data that the user can check on the screen, the first relating to a extremely prejudicial but easy to fix behaviour, idling, the second concerning the driver's penalties, which are also utilized to calculate the score, and finally the charts, which also include a behaviour which isn't present in the previous data, the practice of a constant speed while driving. All of this was created based on the eco-driving analysis explained in the beginning of the second chapter.

3.3. Screen Layout

The module's main screen can be seen in figure 2. Accessed in the application navigation bar to the left, it shows the drivers' journey data on a simple layout with reduced user interaction. Its elements are: 1) Driver and Journey identification; 2) A Spinner used to change the selected journey. By default when navigating to this screen, the latest journey's data will be shown, but the user can switch to any other recent journey or see data regarding multiple journeys done in a certain day; 3) A Timestamp showing the last update time, meaning the last time the system received new eco-driving data regarding the driver's journeys. If the user is currently on this screen when new data arrives, the screen (and timestamp) are automatically refreshed; 4) A help button that opens up a small guide explaining

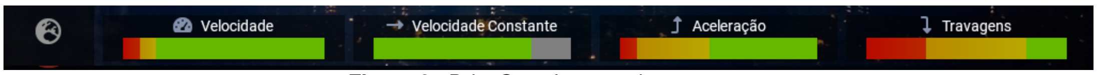


Figure 3: *DriveGreen*'s secondary screen.

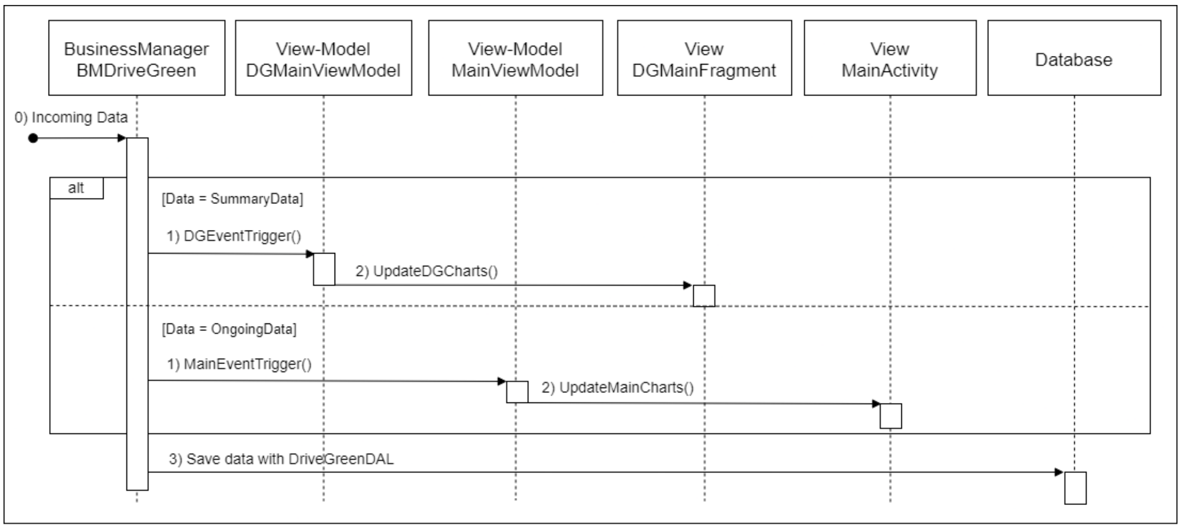


Figure 4: New journey data sequence diagram.

the module's features and purpose; 5) The overall journey data such as total time, distance, engine time, idling time and fuel consumption and efficiency; 6) Penalty counters, regarding speed, acceleration, deceleration, idling, engine temperature and rotations; 7) Charts evaluating the drivers' performance regarding speed, acceleration, deceleration and constant speed; 8) The journey score; 9) And eco-driving suggestions.

The secondary screen corresponds to a set of the already explained four charts displayed in a bar at the bottom of the application's screens during ongoing journeys. This can be seen in figure 3.

This is basically part of the overlay surrounding the several application screens, which consists of the top toolbar, the navigation bar to the left, and this new bottom bar. These charts are automatically hidden whenever the user visits the module's main screen, when there is no ongoing journey, or when filling complex forms in other screens.

3.4. Business Layer

In this section we take a look at the module's business layer, basically the main classes and interfaces needed to save, transform and show data within the module. The protocol regarding the communication of this data to the module is covered in the next section.

Firstly we have *DriveGreenData*, a Model class which defines the objects that hold the journeys' data, which we also call a eco-driving *report*. The reports attributes consists of the driver's and journey's identifiers, the journey's starting time and a list of objects of the *DriveGreenVariable* class, which holds all the data measured during a journey. This simple class contains only two attributes, an unique identifier corresponding to the variable being measured, and of course its value. There's also a *DriveGreenDB* class, which is almost identical to the first one, except that its variables list is now a string, since as the name indicates, it is the class used for holding the application's database objects. Since the database uses SQLite, the list

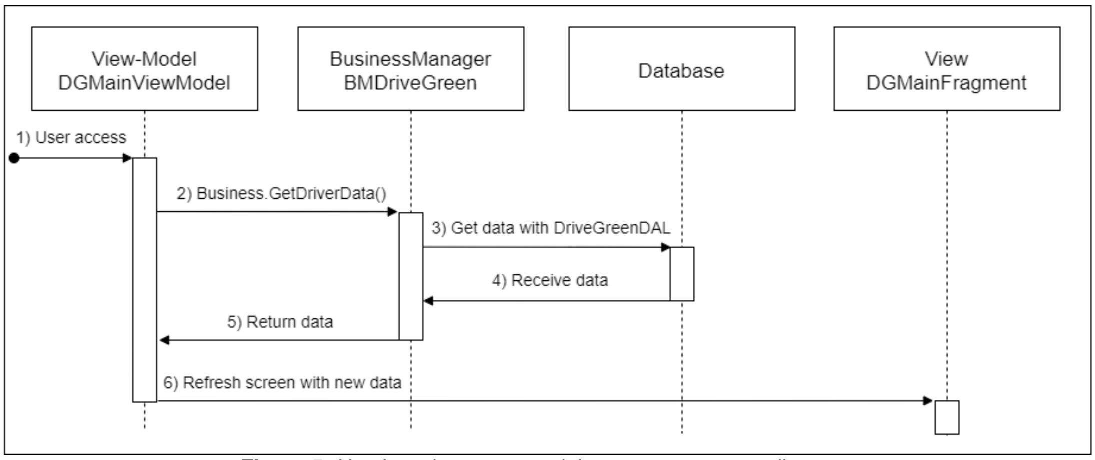


Figure 5: User's main screen module access sequence diagram.

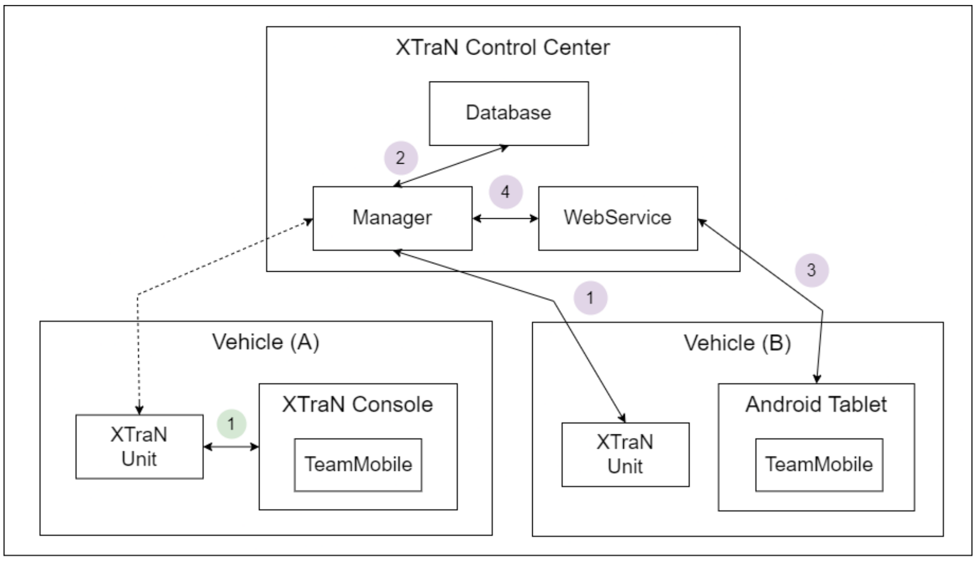


Figure 6: *DriveGreen* communication protocol architecture.

has to be serialized to a string.

To access this database we have a *DAL* (Data Access Layer) class, in which we define the methods which are essentially the queries ran in the database. Regarding *DriveGreen*'s context, we want methods that fetch the eco-driving reports based on the current driver's ID or a specific journey based on its codes (IDs).

We also make use of *event* classes, a C# property used to send notifications and/or data to other classes [17]. When the module receives new journey data it fires these events in order to update elements that are currently visible on the screen. For ongoing journey data there is an event to update the bottom bar charts, and for summary journey data there is another one to update the module's main screen if the user is currently using it.

There is also a service, utilized for communicating with the new web service's API, which is covered in the next section, but the most important and more complex component is without a doubt the controller, which manages the interaction between all the classes, the *DriveGreenBusinessManager*.

We can consider two main operations or communication flows within this internal part of the module's business layer. The first one, regarding the process when receiving new journey data, can be seen in the sequence diagram in figure 4.

The module starts by receiving new journey data (0), parsing and interpreting it. Again the details regarding this topic are explained in the next section. Now the controller, having this data already transformed to the *DriveGreenData* class sends it to the corresponding View-Model (to the module's VM if receiving summary journey data or to the main application's VM if receiving ongoing journey data) by firing an event (1). The View-Models then update the corresponding views (2). Besides updating the screen information, the data is also saved on the application's database, using the mentioned DAL class (3).

A second sequence diagram, seen in figure 5, can be considered regarding the user's access to the module's main screen (1). The View-Model triggers the *GetDriverData()* method from the controller (2), which again uses the *DAL* class to retrieve the necessary data (3,4). The controller returns the data to the View-Model (5) who updates the corresponding View (6), automatically refreshing its screen.

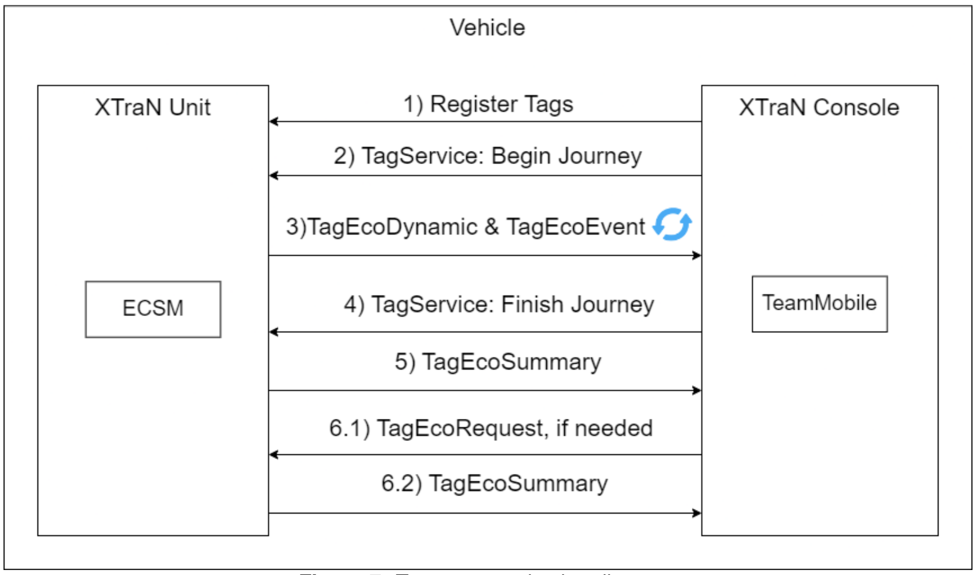
3.5. Communication Protocol

The module's communication protocol in order to receive journey data is composed of two different parts. Direct communication with the *XTraN* unit through *Tags*, and retrieving data from the central database using a new web service. The first method requires the application to be hosted on a *XTraN* on-board console, while the latter occurs when using a regular Android tablet. This can be seen in figure 6, which will be referred to in the coming sections.

3.5.1 Tags

The Tags are used to exchange information directly between the application and the *XTraN* unit (figure 6 - connection 1, green). These Tags objects which are interpreted in a particular way by the *XTraN* formatter and parser. When booting up, the application sends a message to the *XTraN* unit to register the Tags it wants to periodically receive.

The module uses five different Tags: 1) *TagEcoDynamic* where the information regarding ongoing journey data is sent to the application, used to up-


Vehicle
XTraN Unit
XTraN Console
1) Register Tags
2) TagService: Begin Journey
3)TagEcoDynamic & TagEcoEvent
ECSM
TeamMobile
4) TagService: Finish Journey
5) TagEcoSummary
6.1) TagEcoRequest, if needed
6.2) TagEcoSummary
Figure 7: Tags communication diagram.

date the bottom bar charts; 2) *TagEcoSummary*, where journey summary data is sent to the application, used to update the module's main screen; 3) *TagEcoEvent*, which notifies the application of new events regarding the discussed penalties; 4) *TagEcoRequest*, sent to the *XTraN* unit to explicitly request a *TagEcoSummary* update; 5) and *TagService*, sent to the *XTraN* unit to notify it of the journey's current status (ongoing/finished).

We can see this communication diagram in figure 7.

While *TagService*, an older application Tag, utilizes common attributes in its class such as bytes and strings (before being converted into byte arrays), the other four new Tags use Google's *Protocol Buffers* [18]. *Protobuf* is a format utilized to create, define the structure and serialize data messages. It is first created a definition file, much like a *JSON* file, which is then converted to a class file in the desired language, in this case C#. This file makes it so we can define these data types within our system, all encased inside a single byte array object.

3.5.2 WebService

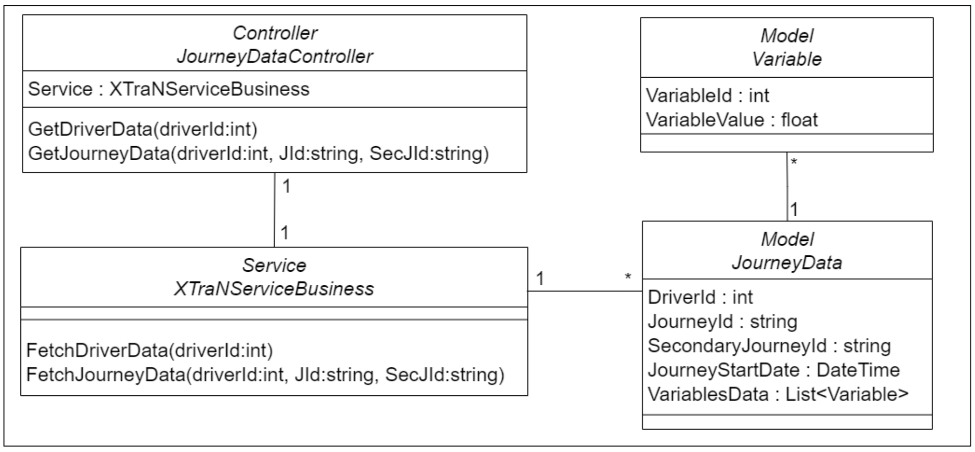
Keeping in mind the diagram in figure 6, when it isn't possible to communicate directly with the *XTraN* unit, the unit sends the journey data to the company's central database. This goes through a *TCP* channel (connection 1, purple) and reaches a Manager component which mediates the database access (connection 2, purple). In order for the application to retrieve the needed data, we need a new web service to send API requests to (connection 3, purple), that then accesses the Manager (connection 4, purple) in the company's server.

The web service uses a REST architecture, developed using C# and *ASP.NET Core*. Its main API, regarding the journey data, can be seen in a diagram in figure 8. The controller, who receives the API requests, has two access points, one that obtains all the recent journey reports of a certain driver, and another that retrieves a specific journey report. The controller then communicates with the *XTraNServiceBusiness* who is the one accessing the Manager and retrieving the requested data in *JSON* format. We can also see the data model classes, identical to the ones presented before in the Business Layer section.

On the application's end, there's a *DriveGreenApiService* class which contains the necessary attributes (the *HTTPClient* and base URL) and methods (to access the mentioned endpoints) to execute asynchronous calls to the web service, and transforming the *JSON* back to the business layer's model class objects.

Additionally, there is a second API, associated with a variable catalog. The web service maintains a small in-memory database table with a variable catalog, which is a record of the different types of variables regarding journey data that are present at any time in the main central database. The catalog consists of multiple *VariableCatalog* model classes, each associated with a unique ID, name and description, and its measured unit, again with an ID, name and description.

The application can access this secondary API to retrieve the updated variable catalog, which makes it easier to update variable parameters in the user's screen. For example, if we consider that each displayed journey variable, such as the distance, has an associated text box for its value and another for the measurement unit, if the vari-



Controller
JourneyDataController
Service : XTraNServiceBusiness
GetDriverData(driverId:int)
GetJourneyData(driverId:int, JId:string, SecJId:string)
1
1
Service
XTraNServiceBusiness
FetchDriverData(driverId:int)
FetchJourneyData(driverId:int, JId:string, SecJId:string)
1
*
Model
Variable
VariableId : int
VariableValue : float
*
1
Model
JourneyData
DriverId : int
JourneyId : string
SecondaryJourneyId : string
JourneyStartDate : DateTime
VariablesData : List<Variable>

Figure 8: Web service's journey data API.

able catalog unit description for that variable is changed, we can automatically update this information on the module's screen.

4. Evaluation & Testing

Throughout the project's development multiple tests were done to assure its quality. The module's functionality tests were done using an Android emulator, simulating received journey data and making sure the system behaves as it should. Regarding the external communication protocol, tests will be done using an *XTraN* unit and console, as well as tests concerning the interaction with the web service.

The initial goal was to have the end users perform usability tests, evaluating the module's behaviour and impact in their work and driving style. However this ended up not being feasible. The application's users were not expected to take their time after a day's work to answer our questionnaire, and Tecmic wasn't able to provide any type of incentive since these users aren't their direct clients. With this in mind, the planned usability tests will be performed by a small team of Tecmic's developers. Later on, when a beta version of this new application update is released, it will be made available to some of Tecmic's clients. This is the time when some feedback can be gathered, coming from the designated employee in each of these client companies.

5. Conclusion

In this chapter we present some conclusions about the development of this project, and refer future work.

5.1. Conclusions

While the testing and release of this module isn't completely finished yet, we can still make some conclusions about its development.

The project's goal was to add a new eco-driving support module to the *TeamMobile* application, used by company drivers belonging to Tecmic's clients. The focus was on a simple, clean layout that helps the user improve his driving style and these goals seem to have been achieved. The application now features a bottom bar with four different charts that measure the driver's performance on the ongoing journey. This layout is small and non intrusive and is hidden whenever necessary, that is, when the user is filling work-related forms in other screens or when there isn't any journey currently selected. The module also provides a new main screen in which drivers can see the data about a finished journey, or even the summary of all the daily journeys, associated with a score and a small list of suggestions to improve driving habits. From a technological point of view, we added a new mechanism to the communication between the vehicle's *XTraN* unit and the mobile application, the *Protocol Buffers*, which simplify the creation and interpretation of messages between them. We also developed a new Web service that makes it easier for the application to communicate with the control center without having to rely on the *XTraN* unit.

5.2. Future Work

In a short-term point of view, the future work consists in finishing the module's external communication protocol testing phase, passing on to the usability testing done by the company and releasing a beta version of the application to some of its users. Based on the feedback received from these tests, some final adjustments will be made, which will ultimately lead to the official new application version release.

However there also some other planned developments: a) The improvement of the score and suggestions algorithm. Besides increasing the amount of available suggestions we want to associate these two components with *gamification* elements, in order to improve user engagement. The score and suggestions should reflect on the driver's progress, and his effort to fix the recommended behaviours; b) Adding features that allow the creation of eco-driving reports on Tecmic's Web platform. This will be available to the drivers as well as their supervisors.

References

[1] INGTECH, "What is eco-driving? - INGTECH," May 2020. Last accessed 30 Oct 2022.

[2] Y. Huang, E. Ng, J. Zhou, N. Surawski, E. Chan, and G. Hong, "Eco-driving technology for sustainable road transport: A review," *Renewable and Sustainable Energy Reviews*, vol. 93, pp. 596–609, 2018.

[3] S. Innamaa and M. Penttinen, "Impacts of a green-driving application in city buses on fuel consumption, speeding and passenger comfort," *IET Intelligent Transport Systems*, vol. 8, no. 5, pp. 435–444, 2014.

[4] J. Tulusan, T. Staake, and E. Fleisch, "Providing eco-driving feedback to corporate car drivers: what impact does a smartphone application have on their fuel efficiency?," in *Proceedings of the 2012 ACM conference on ubiquitous computing*, pp. 212–215, 2012.

[5] Tecmic S.A., "Gestão Profissional de Transportes de Passageiros - TECMIC," Apr 2021. Last accessed 30 Oct 2022.

[6] Samsara, "Eco Driving - Samsara," Jan 2022. Last accessed 30 Oct 2022.

[7] INGTECH, "ECO DRIVING - Intelligent Fleet Management Solutions - INGTECH," Jan 2022. Last accessed 30 Oct 2022.

[8] 0vZ, "Car Scanner ELM OBD2 – Apps on Google Play," Jan 2022. Last accessed 30 Oct 2022.

[9] East Software Coders, "ECO-Driving Speedometer - Apps on Google Play," Oct 2021. Last accessed 30 Oct 2022.

[10] Hayandroid, "Drive Eco - Apps on Google Play," Nov 2021. Last accessed 30 Oct 2022.

[11] Ecologic.io, "Ecologic - Apps on Google Play," Dec 2021. Last accessed 30 Oct 2022.

[12] I. Hawkins, "Torque Pro (OBD 2 & Car) - Apps on Google Play," Jan 2022. Last accessed 30 Oct 2022.

[13] Palmer Performance Engineering, "DashCommand (OBD ELM App) - Apps on Google Play," May 2021. Last accessed 30 Oct 2022.

[14] Microsoft, "What is Xamarin? - Xamarin - Microsoft Learn," Sep 2022. Last accessed 30 Oct 2022.

[15] MvvmCross, "Getting Started with MvvmCross - MvvmCross," Sep 2022. Last accessed 30 Oct 2022.

[16] TechTarget, "What is Model-View-ViewModel (MVVM)?," Feb 2019. Last accessed 30 Oct 2022.

[17] V. Pecanac, "Events in C# - Code Maze," Mar 2022. Last accessed 30 Oct 2022.

[18] Google, "Protocol Buffers - Google Developers," Oct 2022. Last accessed 30 Oct 2022.