DEPARTMENT OF ELECTRONICS,
TELECOMMUNICATIONS AND INFORMATICS

8240 - INTEGRATED MASTER'S IN COMPUTER AND
TELEMATICS ENGINEERING

V2X: Medical Emergency Service 2.0

Authors:
Lúcia Sousa 93086
Manuel Couto 93285
Rafael Dias 95284
Raquel Pinto 92948
Rodrigo Martins 93264

Supervisors: Susana Sargento, Pedro Rito, Miguel Luís
Collaborators: Francisco Castro, João Fernandes

PEI 2020/2021 Semester 2
6 July 2021

Report on the Project of Informatics Engineering Project of the Telematics and Computer Engineering Integrated Masters authored by Lúcia Sousa, Manuel Couto, Rafael Dias, Raquel Pinto, Rodrigo Martins under the guidance of Susana Sargento, Pedro Rito, Miguel Luís, Full Professors at DETI, - University of Aveiro, and Bosch's collaborators Francisco Castro, João Santos

Abstract

Vehicle-to-Everything communications are an innovative and emergent technology that allows the exchange of cooperative messages, alerts, and multimedia in a wireless vehicular environment. It can connect not only vehicles to other vehicles but also to infrastructures, cellular networks, cloud, and vulnerable road users, such as pedestrians. This way, it covers a vast number of applications from traditional infotainment to the most advanced functions of cooperative and assisted driving.

This project aims to help reduce the response time of emergency services whenever an accident occurs, using Vehicle-to-Everything enabled vehicles and gateways with cellular communications. Using this communication method, all vehicles hold constant communication, broadcasting crucial information, such as their location or if they suffered an accident, to nearby vehicles. When an accidented vehicle is detected, the gateway vehicle (vehicle responsible of notifying the emergency services) assembles and sends this information to a dedicated dashboard handled by emergency service operators. By using the Web Application, it is possible to receive the occurred accidents and analyze their data, such as the location of the accident, number of vehicles and people involved, as well as videos and livestreams captured by the gateway vehicle or Roadside Unit. This process occurs almost instantaneously, taking no more than a few seconds from the moment the accident happens until it is displayed on the Web Application. After the arrival of the accident, you can see all the information about it and the emergency services can associate an ambulance and see a suggested route to the accident, you can also see the livestream of the gateway car, the livestreams of the city cameras and cars on the road.

Keywords: Cellular Vehicle-to-Anything Communications, Wireless Access in Vehicular Environments, Dedicated Short Range Communications, Cooperative-Intelligent Transport System, Internet of Things, Road Safety

Acknowledgement

We would like to thank our mentors Susana Sargento, Pedro Rito, and Miguel Luís, other colleagues and teachers, who during this semester helped us by providing a lot of information, constructive criticism, and patience that facilitated the development of this project.

We also would like to thank the Bosh collaborators Francisco Castro and João Fernandes for their availability and help.

Contents

Lists of Figures

List of Tables

Acronyms

CAM Cooperative Awareness Message

CV2X Cellular Vehicle-to-Everything

DENM Decentralized Environmental Notification Message

DOM Document Object Mode

DSRC Dedicated short-range communications

EU European Union

LTE Long-Term Evolution

MQTT Message Queuing Telemetry Transport

OBU On Board Unit

RTSP Real Time Streaming Protocol

RSU Roadside Unit

SQL Structured Query Language

TCP Transmission Control Protocol

UDP User Datagram Protocol

URL Uniform Resource Locator

VANETs Vehicular Ad hoc Networks

V2I Vehicle-to-Infrastructure

V2V Vehicle-to-Vehicle

V2X Vehicle-to-Everything

WAVE Wireless Access in Vehicular Environments

1. Introduction

Nowadays, medical assistance for road accidents sometimes comes late for several reasons. One of them is that the interveners may somehow be incapacitated to the point of not being able to call for help.

In this project, Vehicle-to-Everything (V2X) communication is very important. This is an emerging and innovative technology that enables the exchange of cooperation, alert and multimedia messages in a wireless vehicular environment. This type of communication allows connecting a vehicle not only to other vehicles, but also to infrastructures, the cellular/cloud network and vulnerable road users (such as motorbikes, cyclists and/or pedestrians), thus covering a wide number of applications from traditional infotainment cases to the most advanced cooperative and assisted driving functions. However, in a vehicular environment, not all elements of this ecosystem have access to such technology, and it becomes imperative to create systems that enable cooperation between vehicles with vehicular communication technologies, and at the same time, connection to the infrastructure and vehicles that only have vehicular communications.

In this context, and using current vehicle communication technologies (Dedicated short-range communications (DSRC)/ Cellular Vehicle-to-Everything (CV2X)), Long-Term Evolution (LTE), 5G and computer vision, it is possible to create an emergency message for crashed vehicles, able to instantly inform emergency medical response entities with information about the accident coordinates, number of passengers and vehicles involved, and even provide the competent entities with images and livestream of the crashed vehicles. In turn, emergency services can associate an ambulance to a given accident and follow it in real time. This will allow them to optimise the means used in each situation, as well as the speed with which they are used.

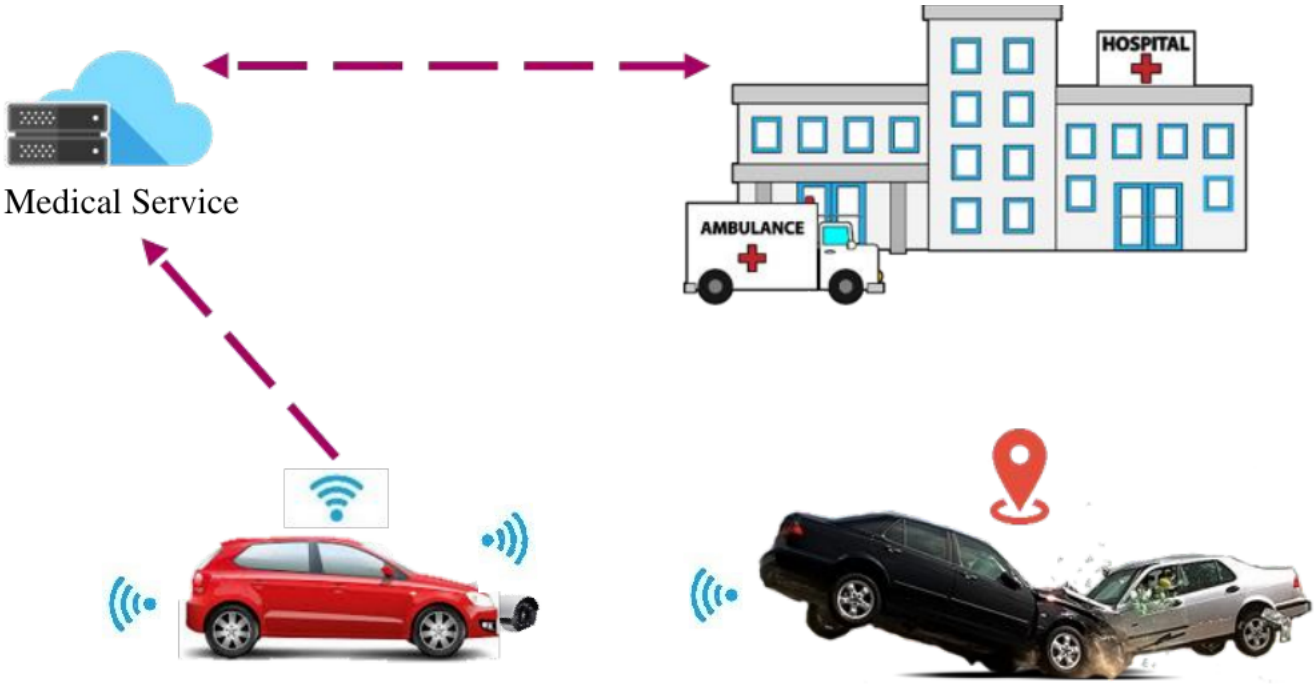

Figure 1-1: Emergency call for crashed vehicles

Figure 1-1 illustrates an example of a situation where the emergency call system for crashed vehicles can be useful. In this figure, a road is illustrated with 2 crashed vehicles, and a vehicle that is driving towards them. The crashed vehicles are equipped with a vehicle communications system (DSRC/ CV2X), which both vehicles use to warn nearby vehicles that they are crashed. In turn, the vehicle heading in the direction of the accident (equipped with V2X communications systems, able to communicate with the cars and the infrastructure), receives the accident warning from the two cars, and starts sending its livestream and recording a video with detailed images of the crashed vehicles. Then, this livestream, this video and the information concerning the crashed cars (coordinates, number of occupants, number of vehicles involved, etc.) are sent to the emergency medical authorities.

Finally, (not illustrated in Figure 1) this information can be used by the road managing authority in question, giving them the possibility to redirect traffic to other adjacent roads. Furthermore, the emergency services can also associate an ambulance to the accident and see the best route to take to the accident, being also able to follow the ambulance in real time. You can also see the cars on the streets and other places with accidents.

The main objective of this project is to develop a system to optimise the way emergency resources are allocated to emergency situations, thus reducing response time, and finally, road traffic management.

2. State of the art

To better understand what was intended in this area for the development of the project, besides familiarising ourselves with the work done the previous year, we read and studied other research works and projects similar to the project that we were going to implement. This made us better prepared for what was ahead of us and made us create new ideas that would be interesting to implement in the project. Some of the most relevant works studied are presented in the following sections.

2.1 Vehicular Ad hoc Networks (VANETs)

Vehicular Ad hoc Networks (VANETs) [1] is a developing technology aimed at providing a safer and smoother travel experience. It is classified into 2 categories:

- **Vehicle-to-Vehicle (V2V) communication**: Enables vehicles to wirelessly exchange information about their speed, location, and heading. Vehicles equipped with appropriate software can use messages from surrounding vehicles to determine a potential accident.
- **Vehicle-to-Infrastructure (V2I) communication**: Is the wireless exchange and bi-directional of data between vehicles and road infrastructure.

This project uses V2V and V2I to create a communication architecture that automatically manages traffic to help emergency vehicles select the least congested route (based on data sent by sensors) to the emergency site. During any difficulty, the system will respond and forward the necessary information on traffic congestion. Furthermore, the system also responds and generates an emergency message at the central node if the emergency vehicle encounters an accident.

2.2 eCall

eCall [2] is a system used in vehicles across the European Union (EU) that, if a vehicle is involved in a serious road accident, automatically makes a free-of-charge emergency call to 112. It can also be activated manually with the push of a button and operates in all EU countries. Wherever you are, if your vehicle is involved in a serious accident, you will be connected to the nearest emergency network, no matter where you bought and registered your vehicle.

When eCall is activated, it connects you to the nearest emergency response center using a phone and data connection. This enables the driver and passengers in the vehicle to communicate with the emergency center operator while a minimum set of data is automatically transmitted (exact location, time of the accident, vehicle identification number, and direction of travel).

To obtain this information eCall has access to the crash system and sensor information from the car. The emergency services can then assess and manage the situation. If the eCall system fails, you receive a warning message.

2.3 5GCar

5GCar [3] is an European project with the goal of ”Developing an overall 5G system architecture providing optimized end-to-end V2X network connectivity for highly reliable and low-latency V2X services, which supports security and privacy, manages quality-of-service and provides traffic flow management in a multi-RAT and multi-link V2X communication system” [4]

Some of the V2X services idealized by this project were:

- **Lane Merger assistance** [5]: Using information from the cars around and video from roadside cameras it’s possible to calculate the needed speed for a vehicle to have a smooth and safe lane merge.
- **See-through sensors**: Using front facing cameras it’s possible for a vehicle behind another to get a view of what’s in front to evaluate, for example, if an overtake is possible.
- **Network assisted vulnerable pedestrian protection**: Using several sources of information such as cameras and other sensors it’s possible to detect vulnerable road users and warn cars about their position to avoid crashes. For example pedestrians crossing the pedestrian crossing.

2.4 Backscatter communication (Boosting Crosswalk Awareness)

This paper [6] has the purpose of reducing road victims by warning cars when a crosswalk is being used. They detect the use of a crosswalk with passive sensors and communicate with cars via vehicular networks to warn the driver, using an app or the infotainment system.

2.5 Conclusion

Doing a similarity analysis we can see that project V2X: Medical Emergency Service 2.0 shares some of these papers’ approaches such as:

- Being a cooperative service
- Trading information via standardized messages
- Improving emergency services’ response time
- Using cellular networks, V2V and V2I communication.
- Using paths for emergency services to go to the scene, they can also see the vehicles near the accident and other accident locations. This way, it is possible to avoid the congested areas.

Although a lot of similarities, our project has some new approaches, like:

- Not relying on an accidented car to contact emergency services
- Using video and livestream for emergency services to see the accident
- The emergency services can associate an ambulance for specific accident.

3. Differences between the base model and the new model

3.1 Base model vs New model

	Base	New
Communication module	Sends CAMs periodically (10 times per second) Useful information about the accident is inside an extra container in the CAM CAMs non standard Sends special CAM when detects an accident	Sends CAMs periodically (10 times per second) The DENM message contains all the relevant information about the accident CAMs and DENMs are standard Sends 1 DENM when detects an accident
Gateway selection algorithm	The cars are evaluated based on the fact that they are behind the crashed car	Takes into account the path taken by all cars involved, the cars will be evaluated based on their point of view during and before the accident, this way the car with better footage will be chosen
Camera submodule	Can't provide livestream to the Web App	Can provide livestream to the Web App

3.2 Improvements in the Web App

The Web App received some new features:

- The possibility of associating and tracking the live location of emergency services and seeing which route the emergency service would take to the accident;
- The tracking of vehicles circulating in Aveiro as shown in Figure 7-12;
- In the accident details page the user can see the live location of the emergency services as mentioned before and see other accidents in the map as shown in Figure 7-12, this feature together with the emergency service route is very usefull to select the best route to the accident and to allow better traffic management;
- The user can also request and view livestreams of cars and street cameras near the accident as shown in 7-11 .

4. Conceptual Modelling

4.1 Functional requirements

For this project, the vehicles involved in the accident, those passing close to the accident and ambulances must be equipped with the vehicle communication system. Furthermore, all vehicles should be configured in such a way that they can transmit live images via Real Time Streaming Protocol (RTSP) and record video of the accident. Ambulances don't need to have cameras, but if they do, it will be possible to see their livestream as well.

The choice of the gateway car must be made based on a gateway choice algorithm.

The camera should have a viewable image of the crashed vehicles, have a field of view to capture the lanes and have RTSP support.

To process this information, the web app must be modified in order to process and display this information of all cars, ambulances associated to accidents and Roadside Units (RSUs), for each of the users (administrative staff and emergency institutions such as medical services, police and fire brigades). This one will also have to know where the cameras of the region where the Emergency Institution are (in this case Aveiro). Since there will be a separation of the information to be shown to each user, he will have to have an account.

This project optimises response time, resource allocation, keeping emergency professionals and other users up to date on any accident and the road traffic management.

4.2 Non-functional requirements

- **Usability:** The web application must be easy to use so that the data is understood as quickly as possible by users;
- **Reliability:** The fact that the system handles emergency cases, has a direct impact on the importance of the system to perform its required functions without failing;
- **Efficiency:** The system must be able to process and display all the information from the accidented car to the web application in 5 seconds or less;
- **Capacity:** The web app must be able to analyze and keep records of several accidents at the same time;
- **Availability:** Since the system is designed to help in emergency situations it must have at least 99% uptime so that no help request is lost;
- **Security:** The web application and the roadside unit's internet interface must be secure and robust against attackers;
- **Recoverability:** In the event of a crash or malfunction, resetting should be easy and painless;
- **Maintainability:** The system and the web application should be easy to maintain and upgrade.

4.3 Actors

The target users of the system are emergency institutions such as police, medical services, and fire/paramedics.

The objectives of the system are to optimise the response to an emergency by displaying important details about the accident, such as its severity, the number of people involved and many other relevant information, and to manage road traffic by displaying a route for each ambulance assigned and accidents occurring elsewhere. This information will be displayed on the Web Application and should be as clear and simple as possible so that the emergency services do not have to worry about it being difficult to understand and focus on the information that is passed through the Web Application. The main actors are:

- **Emergency Institution:** The officers of the Emergency Institution have access to all the information related to the emergency they are supporting, as well as the emergencies about the designated city.
- **Administration Team:** The administration team has access to create, delete and change data and users.

4.4 Use Cases

Figure 3-1 presents the use case model of the whole system, having just one package representing the usage of the web application.

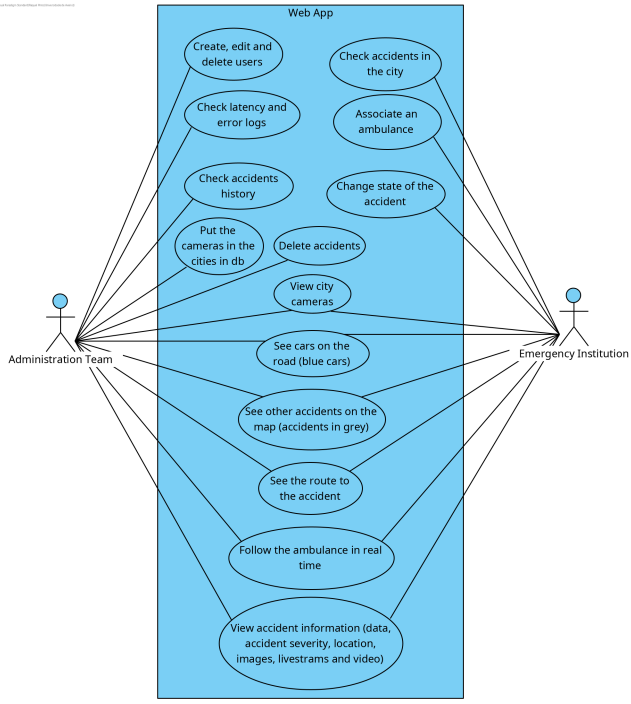

Figure 4-1: Use Cases Diagram

Detailed description of each use case:

- **Create, edit and delete users:** The Administrator can add, remove or edit information of policemen/paramedics or Emergency Control personnel in the database;

- **Check latency and error logs:** The Administrator can view latency and log web app errors in order to fix them later;
- **Check accidents history:** Enables the Administrator to see details of past accidents;
- **Put the cameras in the cities in database:** The Administrator inserts the camera locations of the cities into the database;
- **Delete accidents:** Furthermore, an user with administration permissions is also able to delete accidents from the system;
- **Check accidents in the city:** Users are able to check all the existing accidents in their city, on a map. After selecting an accident, the user will be directed to the corresponding accident page;
- **Associate an ambulance:** Each user can assign an ambulance to a specific accident;
- **Change status of the accident:** Permit users to change the status of each accident in the table;
- **View city cameras):** Users can see the cameras in the city on the map;
- **See cars on the road (blue cars):** Users can see all cars on the road (blue car icon);
- **See other accidents on the map (accidents in grey):** They can also see other accidents on the map (accident icon in grey);
- **See the route to the accident:** All users can see the path of the ambulance associated with a given accident;
- **Follow the ambulance in real time:** Users can follow the ambulance on the map in real time and thus help the ambulance;
- **View accident information:** Allow users to view the accident information (data, accident severity, location, images, livestrams and video).

4.5 Assumptions and Dependencies

To implement this project, there are some dependencies and some assumptions to consider:

- A permanent internet connection is needed for the emergency services to access all the data;
- A server capable enough to handle all the information;
- The hardware can't be damaged when the accident happens;
- The emergency vehicles must be able to send CAM messages to inform where they are located;
- All the vehicles and nearby photoage sources involved (crashed cars, gateway vehicle & street cameras) must be equipped with a vehicular communication system;
- A camera needs to be integrated on the gateway vehicle in order to record images of the accident.

4.6 Deployment diagram

As we can see in Figure 3-2, in the communications module the cars are equipped with On Board Units (OBUs) and ip cameras and the ambulances only with OBUs, which communicate with the web app through RSUs using Wireless Access in Vehicular Environments (WAVE) or if none are present through 4G. The RSUs write the information they receive from the surrounding cars in the mqtt broker.

On the web app side, the web server is able to communicate with a Message Queuing Telemetry Transport (MQTT) broker that is receiving information from the CAMs of vehicles circulating in Aveiro, which allows us to monitor the real time position of the vehicles that are assigned to accidents. The information received from the mqtt broker will be stored in a database influxdb that is optimized to receive information in real time. This allows us to have greater confidence and accuracy in the data received. We also have a Structured Query Language (SQL) database where we store the information about accidents, users, ambulances, congested and some other information as will be explained in the next chapter.

The frontend is done in react and the backend in flask, we also use node.js for the site and the server. The livestream is sent to the server through an RSU or cellular network depending on the situation. The commmunication between the livestream server and the web app backend is done via the SocketIO library.

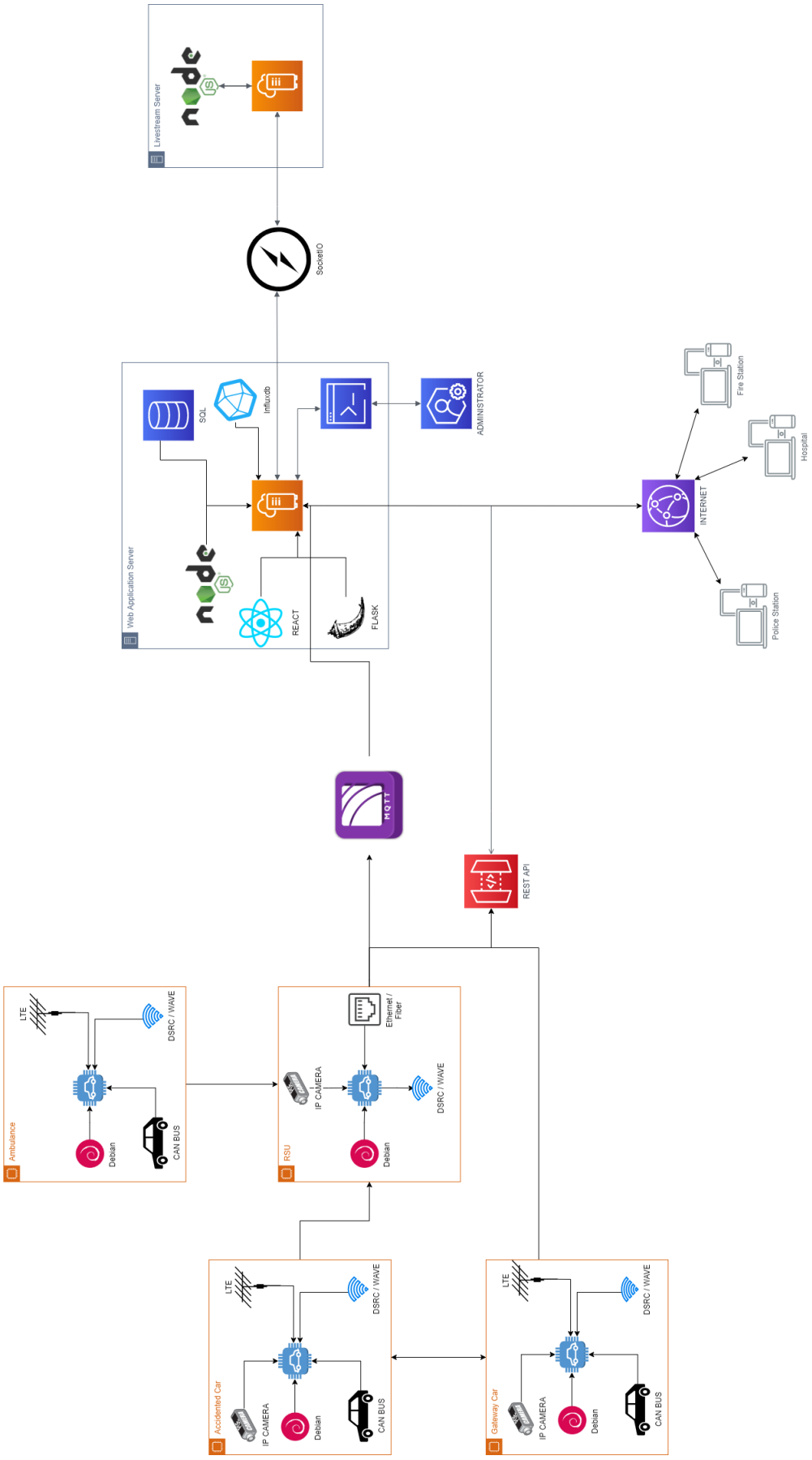

Figure 4-2: Deployment Diagram

5. Procedure

5.1 Communication Module

The vehicles are in constant communication, this is accomplished by using DSRC and the Rendits V2X router to transmit CAMs 10 times per second to all nearby vehicles and DENMs in the case of an accident, both the CAMs and DENMs contain randomly generated information, except for the speed and location which are provided by the GPS.

Figure 5-1: Vehicles in constant communication

5.1.1 V2X Router

The Rendits V2X Router is a platform for research and development in intelligent transportation systems. It supports 802.11p wireless communication and the European Telecommunications Standards Institute ITS-G5 standard. This router was chosen because it is easy to modify or replace parts of the software stack, and to transmit customized messages. In our work we had to modify this router to better suit our needs, this means, we had to create a component inside this router called Vehicle-Service that is responsible to create and send the CAMs and DENMs messages.

5.1.1.1 Vehicle-Service

As mentioned before the Vehicle-Service component is responsible for sending CAMs and DENMs to the nearby cars, both the CAMs and DENMs contain randomly generated information, except for the speed and location which are provided by the GPS. The CAMs are sent 10 times per second and the DENMs is sent when the car detects an accident, but the car doesn't have a sensor that can detect that, so in order to emulate that sensor we made this component listen to a specific User Datagram Protocols (UDPs) port that when triggered sends the DENM message.

5.1.2 CAM Format

> "The standard CAM (Cooperative Awareness Messages) is one of the components of the reference architecture defined by the European Telecommunication Standards Institute (ETSI) for transmitting geographically aware information with relevant data for other vehicles." [7]

Simple CAM

Container:	Bytes:	Data Element:
Header	1	MessageID (=2)
	4	StationID
	4	GenerationDeltaTime
Container Mask	1	ContainerMask
Basic Container	4	StationType
	4	Latitude
	4	Longitude
	4	SemiMajorConfidence
	4	SemiMinorConfidence
	4	SemiMajorOrientation
	4	Altitude
High-Frequency Container	4	Heading
	4	HeadingConfidence
	4	Speed
	4	SpeedConfidence
	4	VehicleLength
	4	VehicleWidth
	4	LongAcceleration
	4	LongAccelerationConfidence
	4	YawRate
	4	YawRateConfidence
(Opt.) Low-Frequency Container	4	VehicleRole

Figure 5-2: Standard Sample CAM

The following image shows what a CAM looks like when it is sent or received:

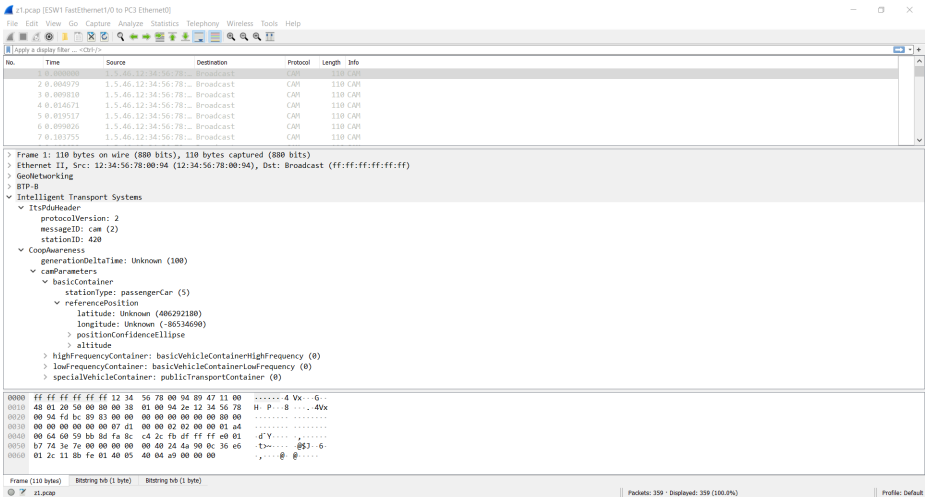

Figure 5-3: Example of CAM message using Wireshark

5.1.3 DENM Format

"DENM (Decentralized Environmental Notification Message) constitutes another type of application support facility providing a notification service about road status." [8]

Simple DENM

Container:	Bytes:	Data Element:
Header	1	MessageID (=1)
	4	StationID
	4	GenerationDeltaTime
Container Mask	1	ContainerMask
Management Container	1	ManagementMask
	4	DetectionTime [1]
	4	ReferenceTime [1]
	4	(Opt.) Termination
	4	Latitude
	4	Longitude
	4	SemiMajorConfidence
	4	SemiMinorConfidence
	4	SemiMajorOrientation
	4	Altitude
	4	(Opt.) RelevanceDistance
	4	(Opt.) RelevanceTrafficDirection
	4	(Opt.) ValidityDuration
	4	(Opt.) TransmissionIntervall
	4	StationType
(Opt.) Situation Container	1	SituationMask
	4	InformationQuality
	4	CauseCode
	4	SubCauseCode
	4	(Opt.) LinkedCauseCode
	4	(Opt.) LinkedSubCauseCode
	0	(Opt.) EventHistory
(Opt.) Location Container	0	LocationMask
	0	(Opt.) EventSpeed
	0	(Opt.) EventPositionheading
	0	Traces
	0	(Opt.) RoadType
(Opt.) Alacarte Container	1	AlacarteMask
	4	(Opt.) LanePosition
	0	(Opt.) ImpactReducationContainer
	4	(Opt.) ExternalTemperature
	0	(Opt.) RoadWorksContainerExtended
	4	(Opt.) PositioningSolution
	0	(Opt.) StationaryVehicleContainer

Figure 5-4: Standard Sample DENM

The following image shows what a DENM looks like when it is sent or received:

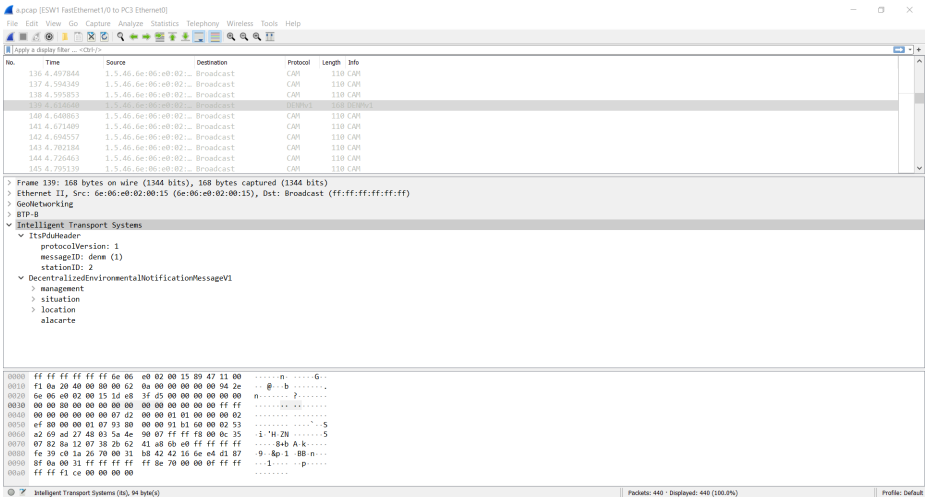

Figure 5-5: Example of DENM message using Wireshark

5.1.4 CAM and DENM Parsing

In order to analyse the data broadcasted, the PCAPPlusPlus library is used to capture the CAMs and DENMs, and to parse them to a C++ program. However, this has raised a problem. The

messages captured were not always CAMs and DENMs and for that reason parse was done to any message that was large enough, a solution had to be found for this.

The solution was to capture all the packets on the wlan0 interface and decode them using the ASN-1 specification, to do this we implemented a python script, sniffer.py, that works like a filter. For all captured CAMs and DENMs from different versions we assigned a 4-byte code, code 1 for the CAMs and code 2 for the DENMs. After decoding the fields, stationID, latitude, longitude, speedValue and headingValue of the CAMs, and the fields stationID, latitude, longitude, numberOfOccupants, externalTemperature and headingValue of the DENMs we build packets to be sent, with the code and values, mentioned above, decoded.

This away it is possible to forward only the CAMs and DENMs through a port, be able to differentiate between them and using the PCAPPlusPlus library, receive these CAMs and DENMs so that they can be analysed in the C++ program.

```
{
    "code",          (4-byte)
    "stationID",     (8-byte)
    "latitude",      (8-byte)
    "longitude",     (8-byte)
    "speedValue",    (4-byte)
    "headingValue",  (4-byte)
}
```

Figure 5-6: CAM packet constitution

```
{
    "code",                  (4-byte)
    "stationID",             (8-byte)
    "latitude",              (8-byte)
    "longitude",             (8-byte)
    "numberOfOccupants",     (2-byte)
    "externalTemperature",   (2-byte)
    "headingValue",          (4-byte)
}
```

Figure 5-7: DENM packet constitution

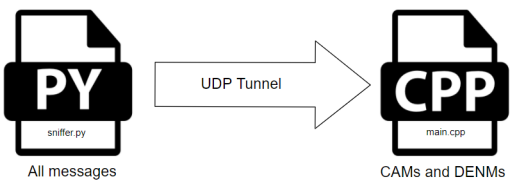

Figure 5-8: Filter of the network packets

5.1.5 The accident

The vehicle-service was implemented to be used to send DENMs on request. This request is made by using a python script, this allows to send a DENM of the accidented car.

The emergency message is sent by the accidented car to the gateway car and from the gateway car to an RSU, if available, if not, the gateway car sends the emergency message via 4G. The information contained in the emergency message was randomly generated, except for the location and speed.

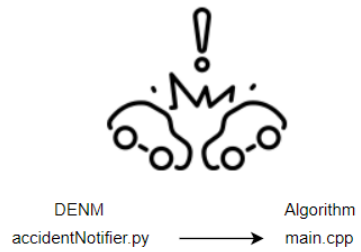

Figure 5-9: Accident simulation

5.1.6 Gateway Selection

Since vehicles are in constant communication, they know where is the accidented car and its neighbors. With this information, each vehicle can calculate which one is in a better position to record the accident, using the position, direction they're headed, and consider also the speed. In other words, each vehicle captures the CAM of the nearby vehicles and runs the algorithm where it compares the entire history of CAM to select the gateway.

Capture of CAMs and DENMs

```
if code==1 then
    #CAM has detected
    CamItems x = {latitude, longitude, heading, speed}
    #cam_container is a multimap, to be able to save
    #CAMs with the same stationID
    cam_container.insert({station_id,x})

if code ==2 then
    #DENM has detected
    DenmItems y = {latitude,longitude, numberOfOccupants,
                   externalTemperature, heading}
    denm_container.insert(station_id,y)
    #the function gateway_selection has as input parameters
    #the multimap with all CAMs and map with the DENM
    gateway_selection(cam_container, denm_container)
    #send DENM to API
    send_to_api(denm_container)
    #send livestream of gateway car
    send_livestream()
    #when the gateway is no longer in a good position
    #to film the accident the livestream is stopped
    stop_livestream()
```

The nearby cars will receive a score, if they have not passed the accident and are driving in the opposite direction. The score is given based on their location in relation to the accidented car, on their heading and their speed. The vehicle with the best score will be the gateway. The smaller the distance to the accidented vehicle, the lower the speed and the lower the comparison between the heading of the accidented vehicle and the heading of the nearby vehicle, the better the score.

The gateway sends the information to the Web App API, this information is extracted from the DENM.

```
{
    "coords": {
        "lat",
        "lng",
    },
    "location",
    "video_id",
    "n_people",
    "temperature"
}
```

Figure 5-10: Message sent by the gateway to the Web App API

Pseudo Algorithm

```
function gateway_selection(CAMs, DENM)
    for CAM in CAMs
        d = distance(DENM.lat, DENM.lon, CAM.lat, CAM.lon)
        comp_heading = comp(DENM.heading, CAM.heading)
    if(pos < 0 && distance >= 10 && CAM.speed > 0)
        s = score(CAM)
        camData.insert({station_id,infoCam})
    camData.sort()
return camData.first
```

5.1.6.1 Test and Results

To test our gateway selection algorithm, we send six CAMs where we have varied the speed, the heading and the location, and one DENM.

The Figure 5-11 illustrates the test done, we sent the CAMs with the IDs 1, 2, 3, 4 and 5, all the vehicles in different positions, from both directions and varying the speed.

With the algorithm running, as soon as the DENM was detected, the vehicles in the worst positions were discarded, for this test, the CAMs discarded were 2, 3 and 4, because they have passed the accident and are driving in the opposite direction, and the rest, 1 and 5, were given a score, as mentioned above, the vehicle in the best position, with the shortest distance and lowest speed got the best score and was chosen as the gateway. The expected result would then be vehicle 1, and in fact it was the one obtained.

The running time of the algorithm was 0.000717772s.

The upload time of the DENM to the API was 0.359925s.

Figure 5-11: Test Scheme

5.1.7 Test a real situation

To test all this in a real situation, we equip the vehicles and the ambulance with OBUs. Their communication with the Web App will be done through the RSUs using WAVE technology.

Through figure 5-12, we can see that vehicles 1, 3 and the RSUs are equipped with IP cameras, and all the vehicles, including the ambulance, are equipped with OBUs. For this specific test, an accident is simulated with a pedestrian crossing the road and being hit by a car.

The accidented vehicle sends a DENMs, which vehicle 1 detects and communicates through the OBUs with the Web App in order to notify it of the accident, as car 1 is the gateway, its livestream is started automatically.

As the RSUs is equipped with a camera, the web app will add the accident and detect that it is able to broadcast your livestream and will start it automatically. After some time the car 3 will approach the accident site and send its livestream also through the RSUs.

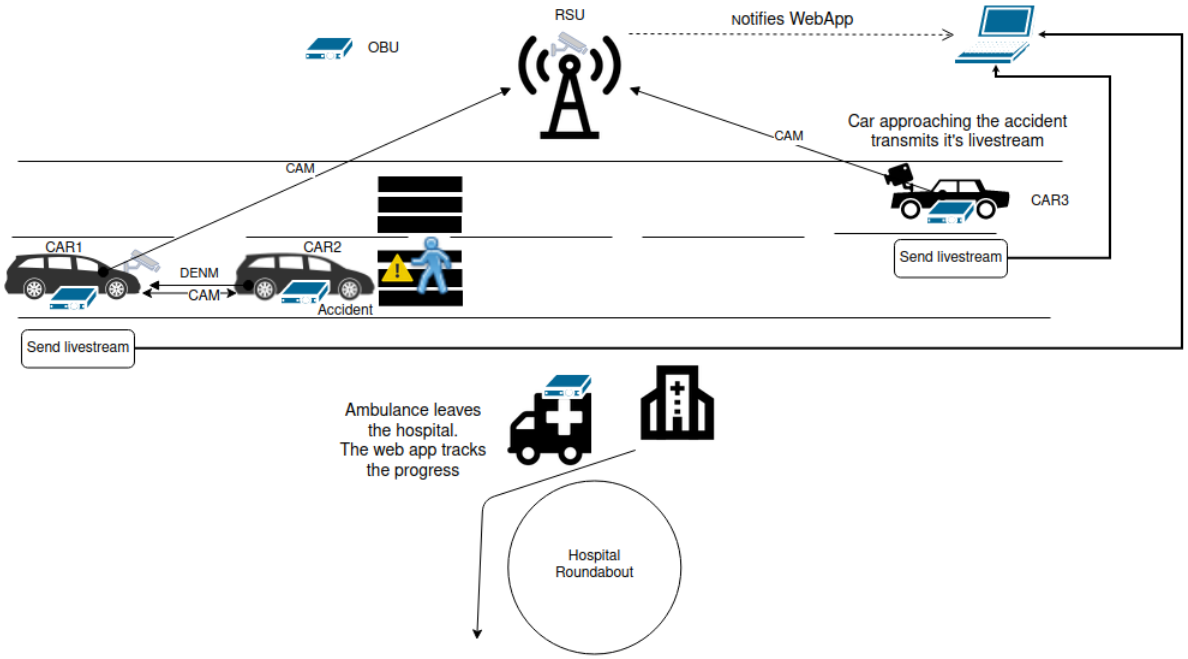

Figure 5-12: Accident Scheme

5.2 Web Application Module

This module is divided in 2 main components: the web site (react frontend and flask backend) and the Livestream Server.

5.2.1 Backend

For the backend we keep using Flask providing a full web server API capable of communicating with a local Database and all the users of the application. Since Flask is a micro web framework written in Python, it allows the use of python Libraries and external APIs and perform Media and Data processing.

We now use InfluxDB which is a time series database designed to handle high write and query loads. Open source server agent to collect metrics from stacks, sensors and systems. Turns any InfluxData instance into a production-ready cluster that can run anywhere.

New features were also created, such as the question of being checked if the car is on fire according to its temperature, and also a function to check if the crashed car is in an area where

there is already the presence of other crashed vehicles in the 500-meter area, if this happens, the database will have information about the cars that are in congested areas.

5.2.1.1 Libraries and external API's

It can be said that Flask is a very much "do it yourself" framework. So, to connect the sqlite database in a Flask application, it was necessary to use flask-sqlalchemy package since there is no built-in database interaction. The only thing needed to do this connection is the Database Uniform Resource Locator (URL).

Another library used was marshmallow to deserialize objects from the database. Moreover, the libraries marshmallow-sqlalchemy and flaskmarshmallow were also utilized. The first one is used to create schemas of the data that comes from the objects deserialization. The second one transforms these schemas into JSON objects to be sent to the application. Summing up, both are thin integration layers for Flask and marshmallow to add more features, such as smooth the data visualization and its transference to the database.

Also, since this project requires the localization recognition of the accidents, the Geopy library was used to make it easy to identify addresses, cities, and countries using third-party geocoders. In this project, the geocoder used was Nominatim.

When it comes to external APIs, the Google Maps APIs was utilized to be possible to embed and retrieve data from Google Maps into the web app. As one of the features of this project is to withdraw images of the roads where the accident happened, the Street View Static API made the job.

We also used node-rtsp-stream and Jsmpeg-player to stream RTSP stream and output to websocket for consumption by jsmpeg.

5.2.1.2 API

Type	Path	Arguments	Values	Required	Optional
POST	/markers/<id>	id	Integer	true	default=0
GET	/markers/<id>	id	Integer	false	default=0
GET	/CarsMarkers/<id>	id	Integer	false	default=0
GET	/CameraMarkers/<id>	id	Integer	false	default=0
GET	/list_ambulances			false	
POST	/addRTSP			false	
POST	/addRTSP			false	
GET	/startStream/<id_acc>/<id_target>	id_acc id_target	Integer Integer	true true	
GET	/stopStream/<accident_id>/<car_id>	id_acc id_target	Integer Integer	true true	
GET	/videos/<id>	id	Integer	false	
POST	/add_accident_denm				
GET	/ambulances_by_user				
FUNCT	checkNearbyCameras()	accCoords	float tuple	true	(location["lat"], location["lng"])
FUNCT	addCamerasToAccident()	cameras accident_id	dictionary Integer	true true	key:camera id value:link rtsp

Table 5-1: APIs Table

FUNCT - function; GET - method GET; POST - method POST.

/markers/<id>

It is used to receive the ambulance id from the 'POST' method, then with the method 'GET' this route gets the localization of the given id.

/CarsMarkers/<id>

Search in the database for cars and then send it to the frontend.

/CameraMarkers/<id>

Get all avaliable cameras and send it to the frontend.

/list_ambulances

Query the database and return all ambulances found.

/addRTSP

Adds the livestream to the database, if the car who send it is gateway it starts automatically.

/startStream/<id_acc>/<id_target>

It is used to Start the LiveStream.

/stopStream/<accident_id>/<car_id>

It is used to Stop the LiveStream.

/videos/<id>

This function check how many lives are associated with the accident and if exists returns the link as a websocket to it.

/add_accident_denm

This is where we add the accident and his details to the database.

/ambulances_by_user

Get all the ambulances avaliable to the current logged user.

checkNearbyCameras(accCoords)

Check if a camera is close enough to the accident to start a stream.

addCamerasToAccident(cameras, accident_id)

Adds available cameras to the accident and starts the stream.

5.2.1.3 Live Location

We implemented a system of live location for all the vehicles we are using, in specific cars and ambulances. In our webapp the user has continuos acess to the real time position of each available vehicle.

While cars are moving they send CAMs to the RSUs, this RSUs reads and parses the information about each car and then stores the parsed information in the mqtt broker.

We are able to subscribe this broker and read from it this usefull information, with that we save the coordinates (latitude and longitude) of each vehicle in our optimized database figure (Figure 5-16: Ambulance and Cars Database).

Then to make this functionality available in our web app we just need to keep querying our database and displaying it on the map.

5.2.1.4 Database

In accident database we added the columns fire, ambulance and congested. Column fire is used to indicate if the car is burning or not (0 = not burning, 1 = burning), column ambulance is used to indicate the number of the ambulance who is attending the accident and column congested indicate if the accident is located in a congested area.

people	n_cars_involved	damage	n_people_injured	video_id	date	status	video_total	city	fire	ambulance	congested
	1	60.0	0	12345678912345	2021-04-21 10:41:07.080769	0	0	Guarda	0	NULL	NULL
	1	85.0	0	12345678912345	2021-04-21 10:43:32.747757	0	0	Guarda	1	NULL	NULL
	2	61.0	0	34324252	2021-04-21 10:56:39.800684	0	0	Castelo Branco	1	NULL	NULL
	1	74.0	0	3432443421	2021-04-21 10:57:40.172800	0	0	Viseu	1	NULL	NULL
	1	49.0	0	3432443421	2021-04-21 10:58:27.631520	0	0	Guarda	0	NULL	NULL
	1	49.0	0	3432443421	2021-04-24 14:56:55.829543	0	0	Guarda	0	NULL	NULL
	1	49.0	0	3432443421	2021-04-24 14:58:23.185347	0	0	Guarda	0	44	NULL
	1	49.0	0	3432443421	2021-04-24 15:24:41.003171	0	0	Guarda	0	NULL	NULL
	1	77.0	0	34324121	2021-04-24 15:26:27.751630	0	0	Aveiro	1	0	NULL
	1	48.0	8	3224121	2021-04-24 15:27:05.667103	1	0	Aveiro	0	44	NULL
	1	67.0	0	3217631	2021-04-28 20:00:01.506623	0	0	Aveiro	0	44	NULL
	1	67.0	0	3217631	2021-04-28 20:08:02.855456	0	0	Aveiro	0	NULL	0
	1	67.0	0	3217631	2021-04-28 20:09:41.148295	0	0	Aveiro	0	NULL	1

Figure 5-13: Accident Database

In this database we associate each user with their avaliable ambulances.

id_ambulance	id_user
12	1
33	1
44	2
45	1
55	3
60	2

Figure 5-14: Ambulance Database

This database is used to link each camera with their RTSP link.

id	latitude	longitude	linkRTSP
1000	40.63476	-8.66038	rtsp://

Figure 5-15: Cameras Database

This database made in influxdb is used to get the live location based on latitude and longitude of the ambulances and the gateway cars, class 10 is an ambulance and class 5 is a car.

time	car_class	car_id	latitude	longitude
1624559130340628000	10	2	40.6344406	-8.6606518
1624608469302721000	5	257	40.6401913	-8.6635557
1624609179871759000	5	274	40.6412623	-8.6534761
1624612614944831000	5	558	40.6360682	-8.6596976
1624627599557482000	5	533	38.7794504	-9.3492336
1624627690491838000	5	1185	38.7794669	-9.3490683
1624628034761681000	5	1068	38.821133	-9.1879178
1624638226633624000	5	472	40.6341231	-8.6599595
1624641915141007000	5	457	40.6340913	-8.6599781
1624643312345873000	5	15	40.635152	-8.6602642

Figure 5-16: Ambulance and Cars Database

This database associates each gateway car with the indicated accident and RTSP link.

name: livestream

time	acc_id	car_class	car_id	link_RTSP	stream_started
1623664630332988000	50	5	423	rtsp://	1
1623685948928079000	49	5	45	rtsp://	1

Figure 5-17: LiveStream Database

5.2.2 Frontend

To best continue last year's project the framework used continued to be React for its nice, responsive and intuitive form. However, React's most appealing feature is its fast rendering performance when using the virtual Document Object Mode (DOM) compared to other frameworks. Since the application is constantly receiving updates, all new information must be fast enough to display those updates.

Since React is a Javascript-based framework, to provide a server environment for the application, Node.js was used. This environment was chosen for its ability to use asynchronous programming, optimizing the performance and scalability of the application, allowing real-time communication and increasing the efficiency of the web application.

In addition, the Sass preprocessor was also used, which stands for Syntactically awesome stylesheets, to make the design process much faster and more efficient.

To get the ambulance path we used the DirectionsRenderer react library (render the route on the map) in combination with the Directions service API from Google (generates the path). The coordinates of the accident are obtained from the SQL database, since this coordinate remains

static over time, the coordinates of the ambulance and the coordinates of the cars on the road (blue cars) are obtained from the InfluxDB database since it needs constant updates.

If users click on the blue cars, they start their livestream which can be viewed on the web app just like other livestreams.

Besides this information, you can also see the locations of the cameras by user location. And just like in blue cars users can start their livestream.

The location of the cars on the road as well as the location of the other accidents allows a better traffic management on the road, so the ambulance can choose another route to the accident.

5.2.3 Livestream Server Component

As mentioned before one of the requirements of this project was the capability of receiving a live image feed from the cars or other sources near the accident. To meet this requirement it was developed a livestream server capable of receiving images from cameras and redirecting them to their respective pages in the web application.

This server was written in NodeJS and connects to the backend of the Web Application via SocketIO. The handling of the frames received from the camera was implemented using the node-rtsp-stream library. The servers sends the images through WebSockets to a specialized video player that displays them. This video player was implemented using jsmpeg-player react component.

The node-rtsp-stream library was chosen because allows for a great costumization of different fields of the livestream, some examples are: the transport mode (UDP or Transmission Control Protocol (TCP)) in this case we used tcp or the resolution of the image. The jsmpeg-player was chosen because has great compatibility with the node-rtsp-stream as the documentation of those libraries mention.

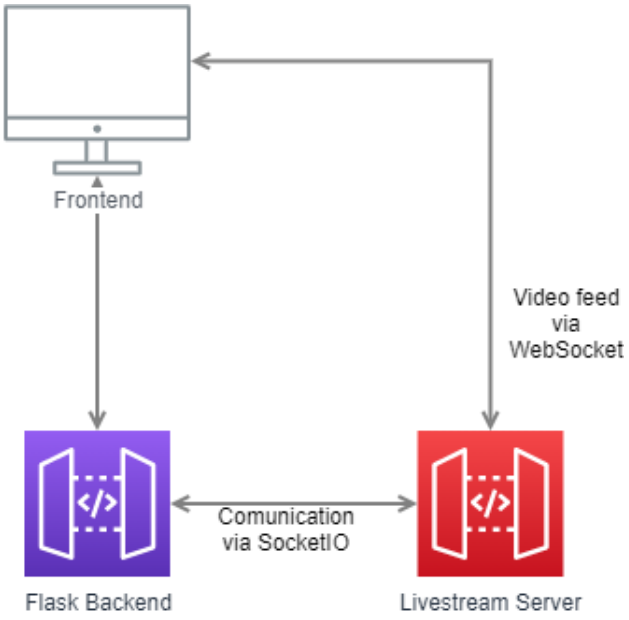

Figure 5-18: Web Application and Server

5.2.3.1 Server Structure

All of the methods inside the server consist in 4 SocketIO endpoints that have a specific role.

- Socket.on "connect": Displays a message to inform that the server as connected successfully to the backend.
- Socket.on "teste": Displays a message to inform that the connection test between the backend and the server was successful.
- Socket.on "rtsp": Used by the backend to make the server start a livestream to a specific camera.
- Socket.on "stop": Used by the backend to make the server stop a specific livestream.

5.2.3.2 Server Message Flow

When a Livestream must start the following message flow happens:

1. The backend sends to the server through the Socket "rtsp" the information about the livestream that must start. This information contains the id of the accidented car, the id of the car or camera and the link of the rtsp stream as exemplified by the Figure 5-19. The backend also stores in the InfluxDB database the information about the livestream as exemplified by the figure 5-20. The `"stream_started"` parameter is used by the backend to know if an the livestream as started so it can send a list containing it to the frontend, '1' means the livestream already started and '0' means the opposite.

```
{
    "accidented_car_id": 43,
    "carID": 21,
    "rtspLink": "rtsp://example@example/11"
}
```

Figure 5-19: Example of a message sent to start a livestream

```
{
    "measurement": "livestream",
    "tags": {
        "car_id": 21,
        "acc_id": 43,
        "car_class": 5
    },
    "time": datetime.now(),
    "fields": {
        "link_RTSP": "rtsp://example@example/11",
        "stream_started": '1'
    }
}
```

Figure 5-20: Example of the information about a livestream stored in the InfluxDB database

2. The server upon receiving this information starts the livestream for that car/camera and sends the video feed through a WebSocket ready for consumption by the video player. The port needed to associate a livestream to a car/camera is calculated by adding 9000 to the id of the car/camera, this is done to ensure that 2 livestreams don't use same port because otherwise the video wouldn't work. The number 9000 was chosen so ensure that the result port wasn't being used. Finally the server stores the livestream object inside a map, the key being the port of the livestream and the value the livestream object itself.

When a Livestream must stop the following message flow happens:

1. The backend sends to the server through the Socket "stop" the information about the livestream that must start. This information contains the port of the livestream that must stop as exemplified by the Figure 5-21. The backend also deletes from the InfluxDB database the information about that livestream.

```
{
    "port": 9021
}
```

Figure 5-21: Example of a message sent to stop a livestream

2. The server upon receiving this information searches in the livestream map for the livestream object using the port as the key. When the object is found the livestreams stops, the port is freed and the livestream is deleted from the map.

6. Message Flow

6.1 Accident detected

When an accident is detected the gateway car notifies the web app by posting a message with the structure of the Figure 6.1 on the corresponding API endpoint (`/add_accident_denm`) described in the section 5.2.1.2.

```
{
    "location": {
        "lat": 40.63480,
        "lng": -8.66040
    },
    "video_id": 60,
    "n_people": 4,
    "temperature": 40
}
```

Figure 6-1: Message sent by the gateway to the Web App API

6.2 Web App handling

Uppon receiving this message the web app reads the information present in the message, stores the accident in the database and searches for nearby street cameras. If there are any cameras in a 50 meter range of the accident this cameras will be associated to the accident and their livestream will automatically start.

To start this livestreams the backend send a message with the structure of the Figure 6.2 through the socketIO endpoint related to this event, this will trigger the Livestream server to start a livestream for the given RTSP link. The structure of this message is the same for starting a livestream of a camera in a car and for a street camera and this is why the parameter "carID" is present in this message, this makes no difference for the livestream server because the implementation for each type of camera is the same.

```
{
    "accidented_car_id": 43,
    "carID": 21,
    "rtspLink": "rtsp://example@example/11"
}
```

Figure 6-2: Message sent by the gateway to the Web App API

After all of this steps the accident appears on the accident table and the livestreams only appear in the associated accident.

While all of this was happening the backend was also keeping track of the passenger cars and ambulances circulating in Aveiro and storing their position in the influxdb as mentioned in section 5.2.1.

7. Results and discussion

7.1 Communication Module Results

This section presents the results of the communication module, and evaluates the performance of our developed software. The result of this module is a program that autonomously detects accidents and sends a livestream and information to the Web App.

To test the performance of the communication module, we made 5 tests to measure the time elapsed between detecting the accident and sending the livestream and the DENMs to the Web App. The test consisted of measuring the time it took from detecting an accident using the DENMs until the livestream is sent, and we measured the time it took until the emergency message reached the Web App.

	DENM Upload (s)	Gateway Selection (s)	Link RTSP Upload (s)
Average time	0.24896	0.0004258246	0.01619988
Standard Deviation	0.04512	0.00004	0.00071
95% Confidence	0.24896	0.0004258246	0.01619988

Table 7-1: Time Measures

- **DENM Upload:** The time it takes, in seconds, for the JSON message to be created and sent via the API until a success message is received.
- **Gateway Selection:** The time it takes, in seconds, since the DENM is detected and the gateway selection process is completed.
- **Link RTSP Upload:** The time it takes, in seconds, for the RTSP link to upload, this process is complete when we receive a success message from the API.

As we can see from Table 7.1, the measured times are within the requirements, the performance of the gateway selection algorithm has been improved, as well as the sending time of the DENMs to the API.

7.2 Live location tracker Results

This section presents the results of the live location tracker component, and evaluates the performance of our developed software. The result of this component is a program that autonomously reads and parses the information available in the mqtt broker and stores it in the InfluxDB.

To test the performance of the location tracker component, we readed and decoded 14 messages about the 3 cars circulating in Aveiro present in the broker and measured the time elapsed between receiving the message, decoding it and storing it in the InfluxDB.

	Decode (s)	Insert (s)
Average time	1,11E-05	0,9836583138
Standard Deviation	0,00001288175384	0,9969841458

Table 7-2: Time Measurements

- **Decode:** The time it takes, in seconds, for the message to be decoded and parsed to a JSON object.
- **Insert:** The time it takes, in seconds, to insert/update the JSON object in the InfluxDB.

As we can see from Table 7.2, the measured times are within the requirements.

7.3 Web Application Results

When both frontend and backend work together, the result is a fully functional dynamic web application.

Looking at the final result, the features provided by this application are:

7.3.1 User View

The login page is the entry page that requires user identification and authentication, performed regularly by entering a username and password combination.

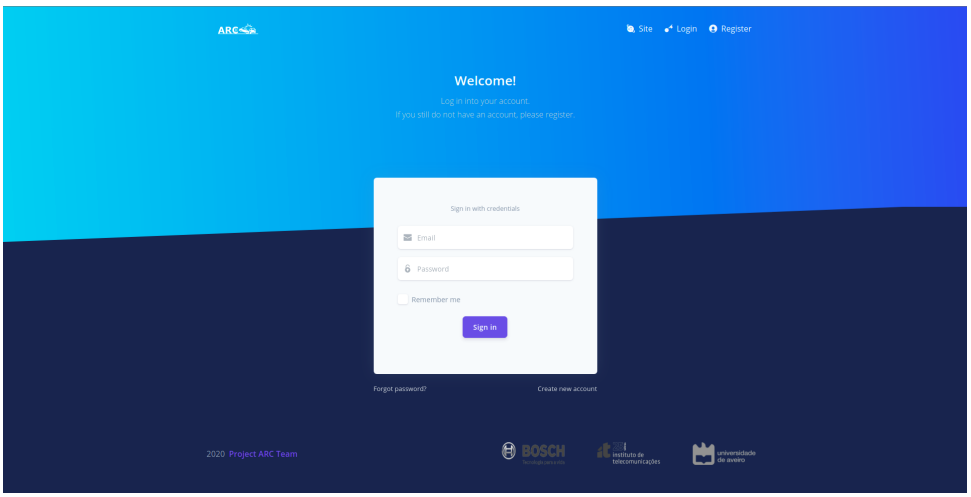

Figure 7-1: Login page from the web app

Whenever an error occurs, the user receives feedback from the web app.

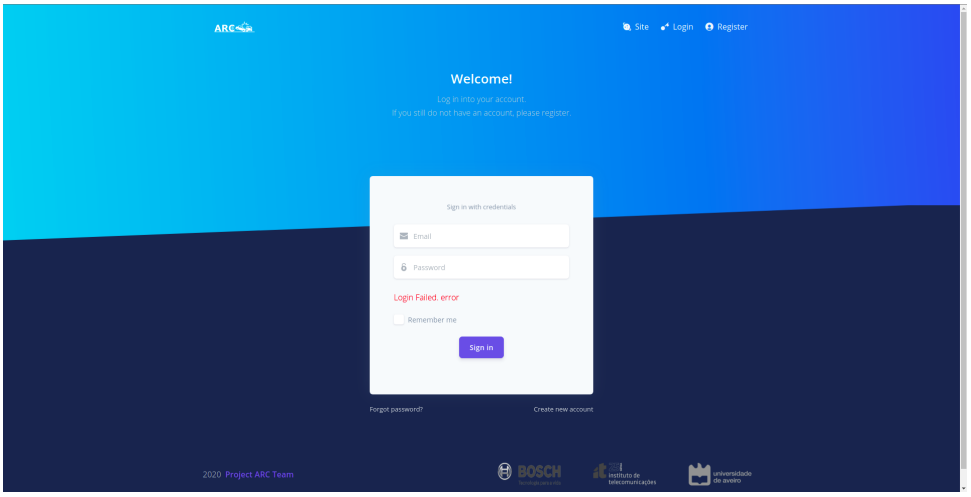

Figure 7-2: Login page with error from the web app

Main page where he can see the statistics regarding all the accidents that happened last month or last week. The user can also see the accidents that happened in some districts.

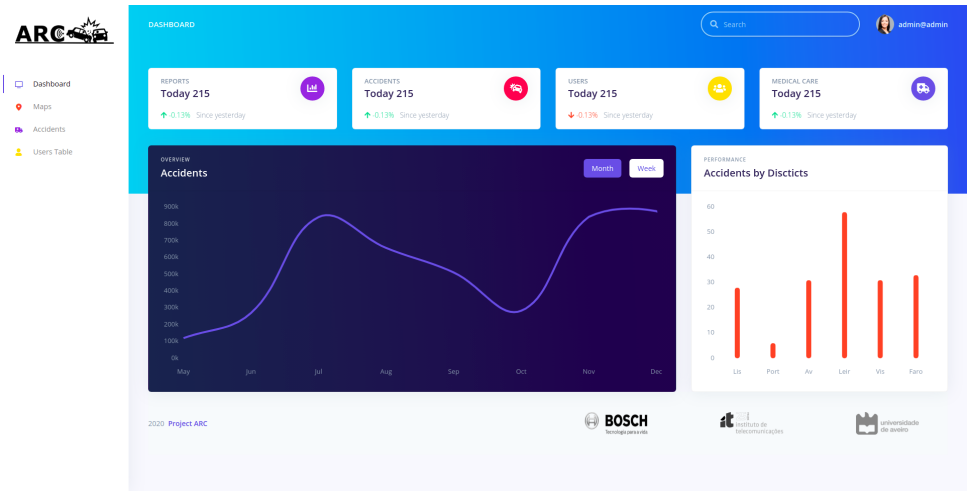

Figure 7-3: Home page from the web app

The ability to look at all the accidents that are in the database on a map with the ability to focus on the user's location.

The colors of the cars represent the actual state of the accident ("Accident still not answered", "Emergency services are on their way" or "Accident resolved"), and it is possible to click on a traffic accident that is being displayed on the map to see its details, and you will be redirected to the accident details page of the corresponding accident.

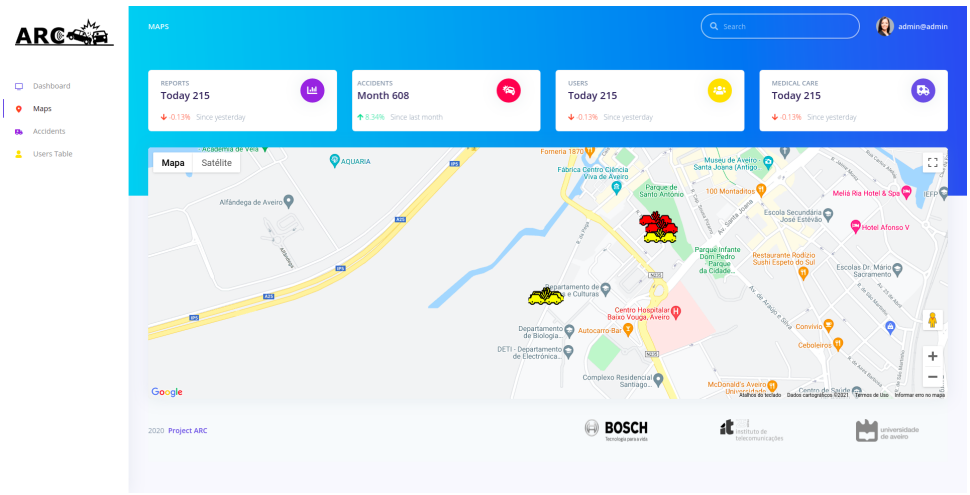

Figure 7-4: Map page from the web app

Checking all accidents in a table capable of filtering these accidents by date and time by date and time, number of cars or people involved, number of injuries, severity, accident status, and fire. It is also possible to to sort the accidents by those that happened today, yesterday, last month, or all in an ascending or descending order. Also, since the user has a city associated with his profile, it is only possible to see the accidents that correlate with his location.

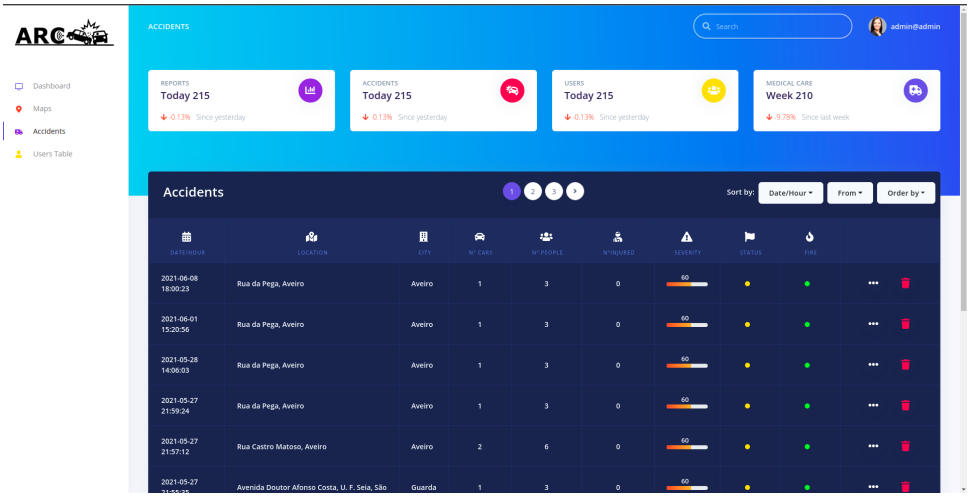

Figure 7-5: Accidents table from the web app

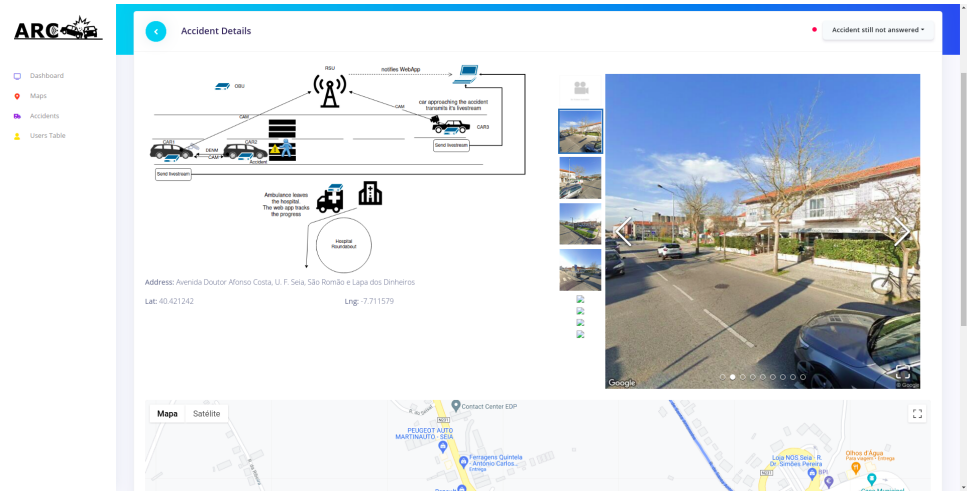

Figure 7-6: Accident details top page from the web app

On this page it is possible to see some information regarding the accident itself, such as its location, a video of the accident, images of the roads where the accident took place, the number of vehicles and people involved, the number of people injured, severity of the accident, livestreams and assigned ambulance. This detail page can also be accessed through the accident table.

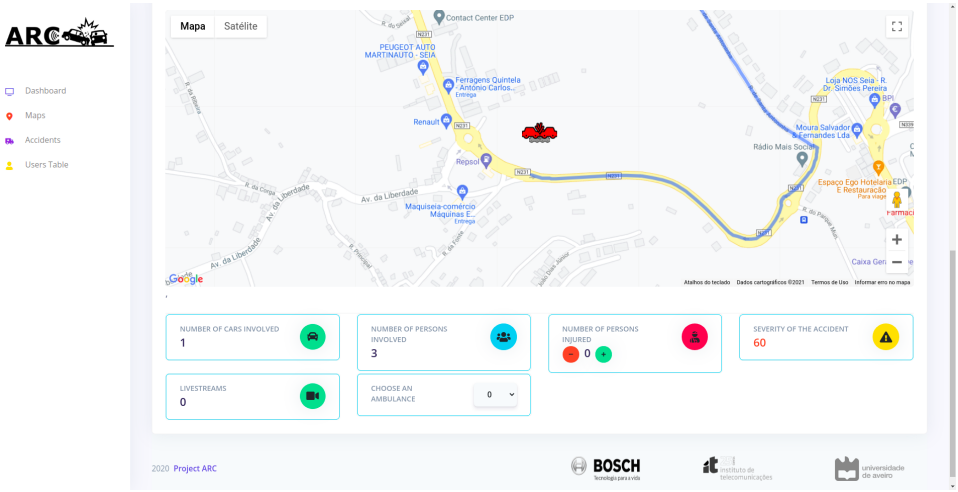

Figure 7-7: Accident details bottom page from the web app

After an ambulance is assigned to the accident in question, it automatically changes from "Accident still not answered" to "Emergency services are on their way".

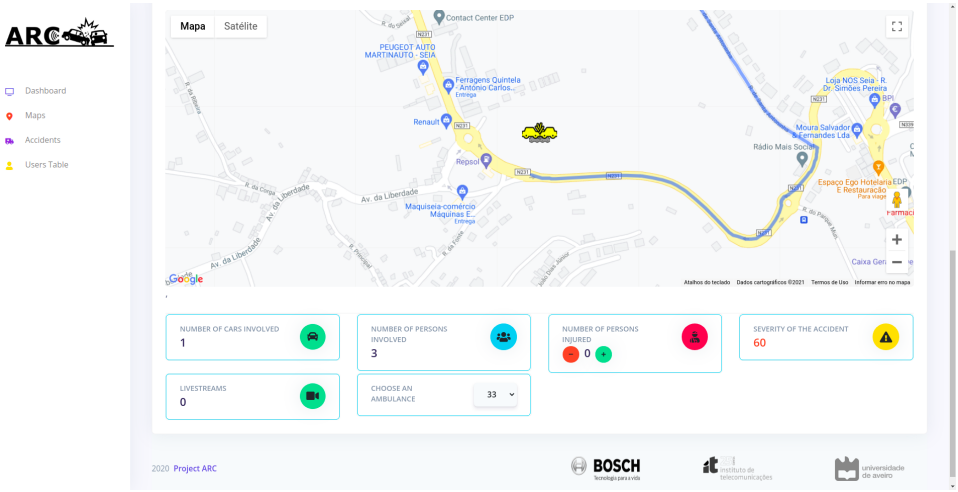

Figure 7-8: Accident details bottom page with ambulance associated from the web app

The user is also allowed to note details about the cars involved in the accident. In addition, the user as the ability to update the number of injured people in case the initial prediction did not match the truth.

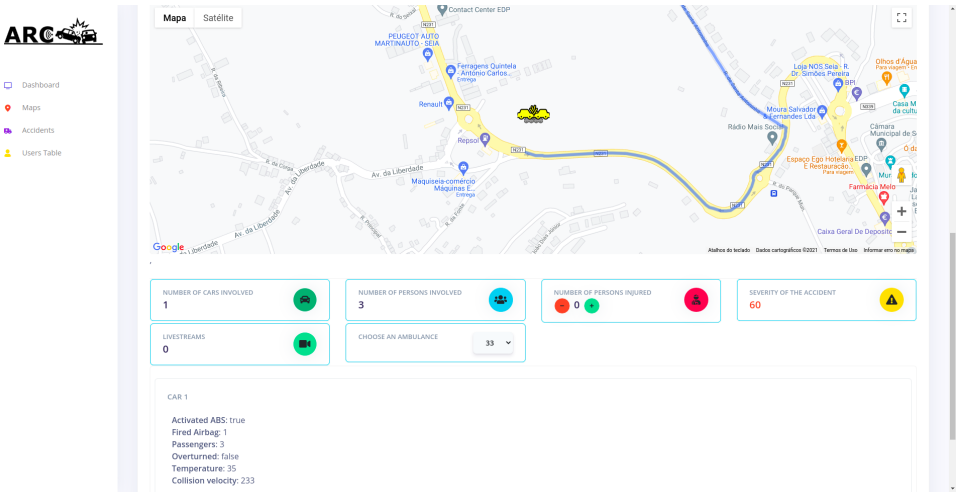

Figure 7-9: Details of the vehicles involved in an accident

Right after the accident message is sent to the webapp, the webapp will display the images from the livestreams appropriate to that accident.

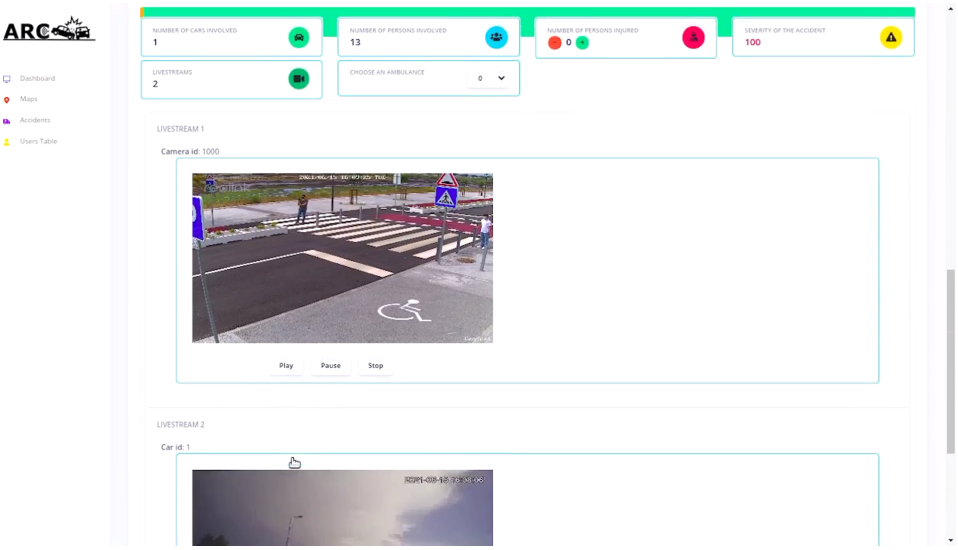

Figure 7-10: Accident details map page with livestream from the web app

The fastest route the ambulance has to take until it reaches the accident site is shown on the map, the remaining accidents are shown in grey and the remaining cars on the road are shown in blue. This makes it easier for the ambulance to reach the scene.

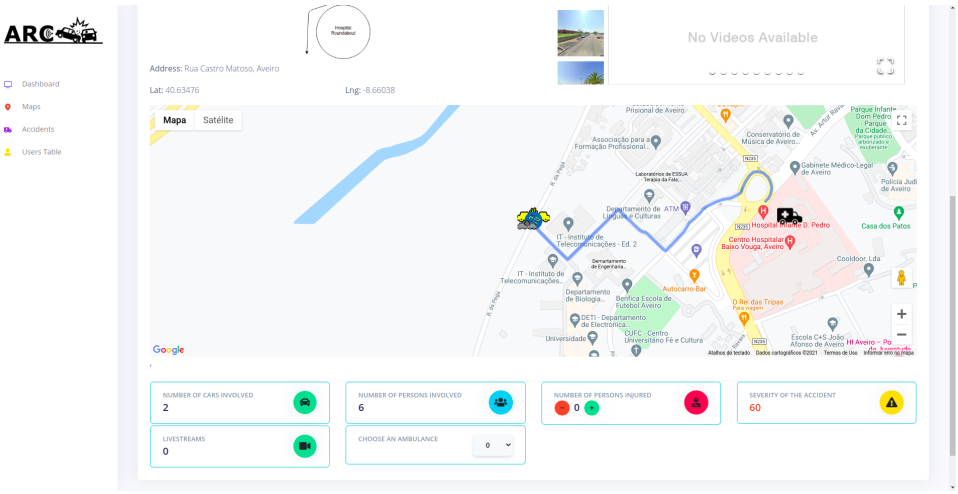

Figure 7-11: Accident details map page with ambulance route from the web app

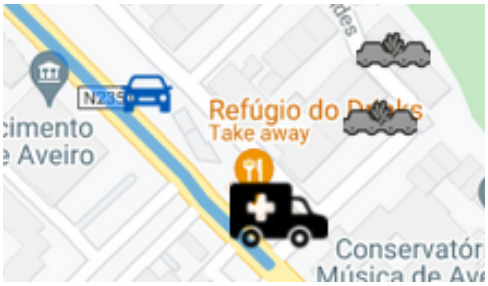

Figure 7-12: Blue cars (cars on the road) and grey cars (other accidents)

The map is always being updated, and after an ambulance is assigned to an accident, it will be possible to follow its route in real time via the map.

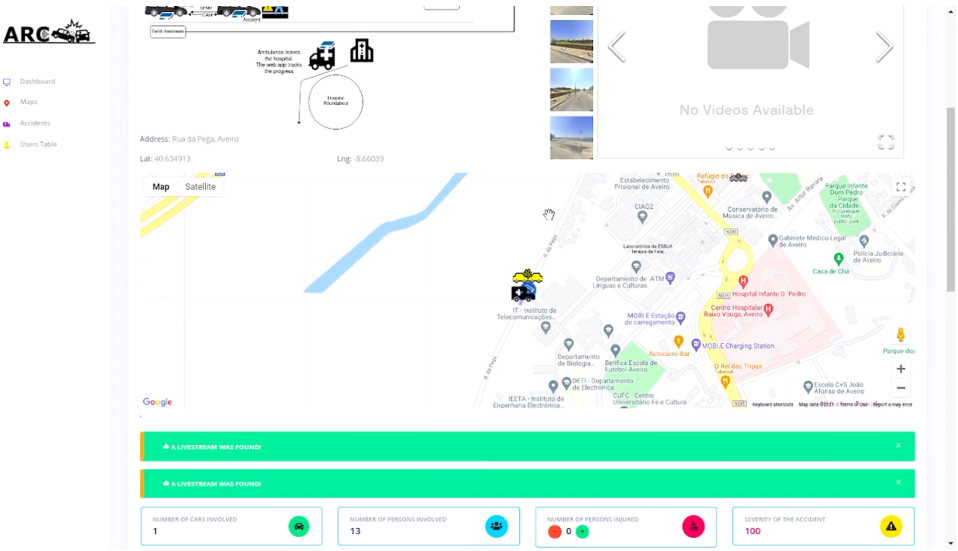

Figure 7-13: Accident details map page with ambulance route in real time from the web app

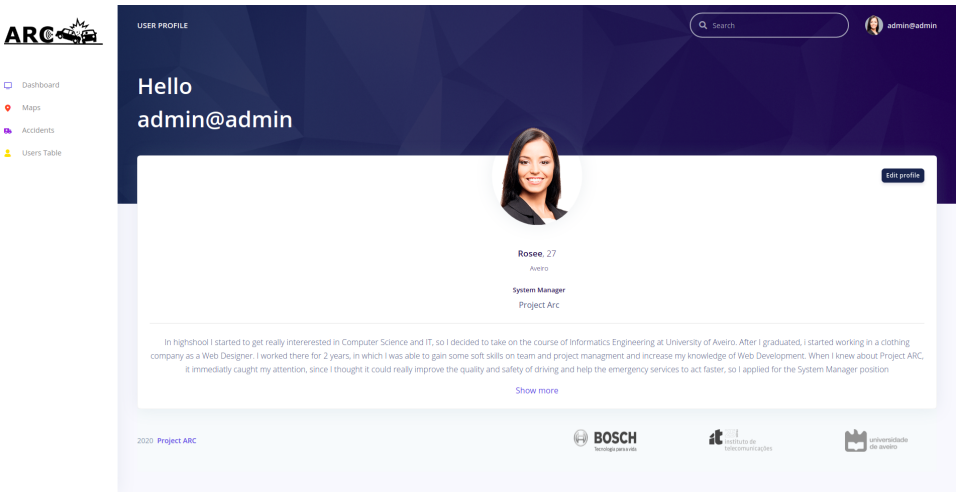

Figure 7-14: User profile page from the web app

You have access to your profile and are able to edit your information. However, information such as e-mail and city cannot be updated unless the user is a system administrator, due to security reasons.

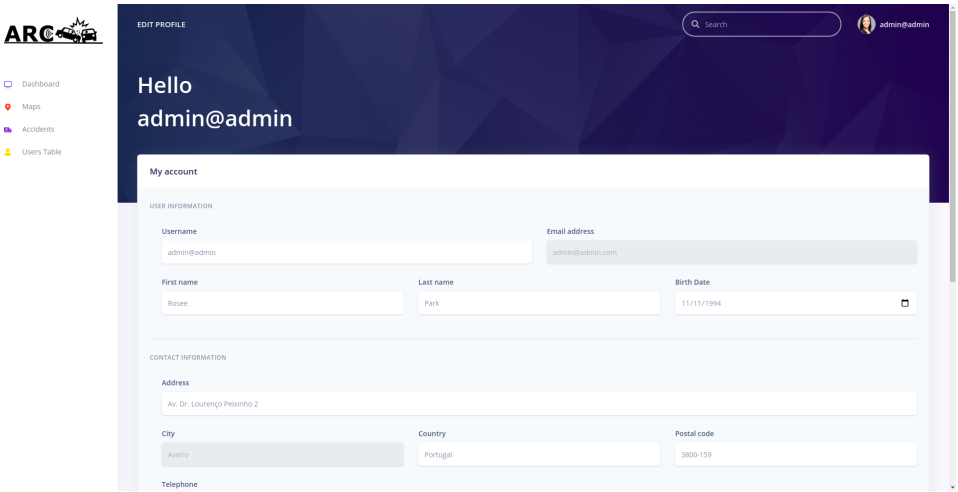

Figure 7-15: Edit profile page from the web app

7.3.2 Administrator View

In case the user is an administrator, the system allows the functionality to manage all user accounts, permissions to add, edit and delete their data. It is also possible for the administrator to create a new user by clicking on the add icon in the users table.

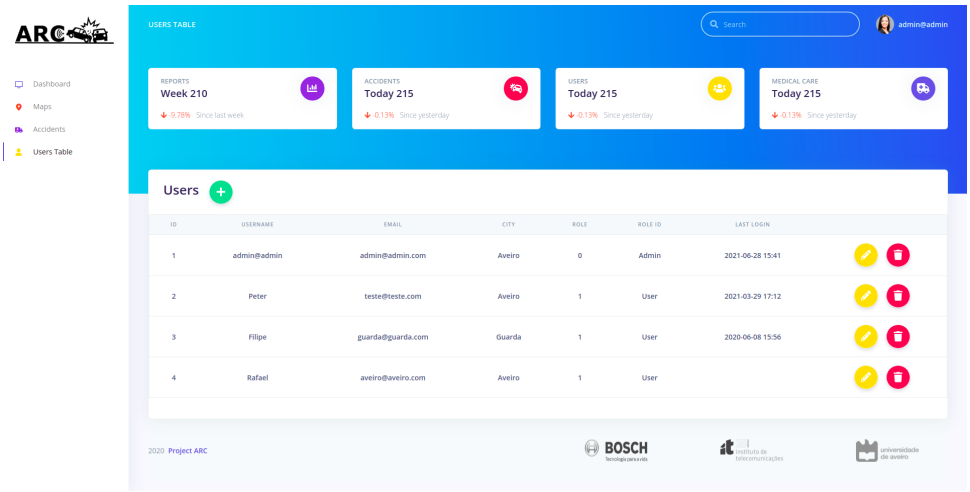

Figure 7-16: Users table with admin view on the web app

The administrator can also delete users or change some information about them, such as username, e-mail, city, role and role id.

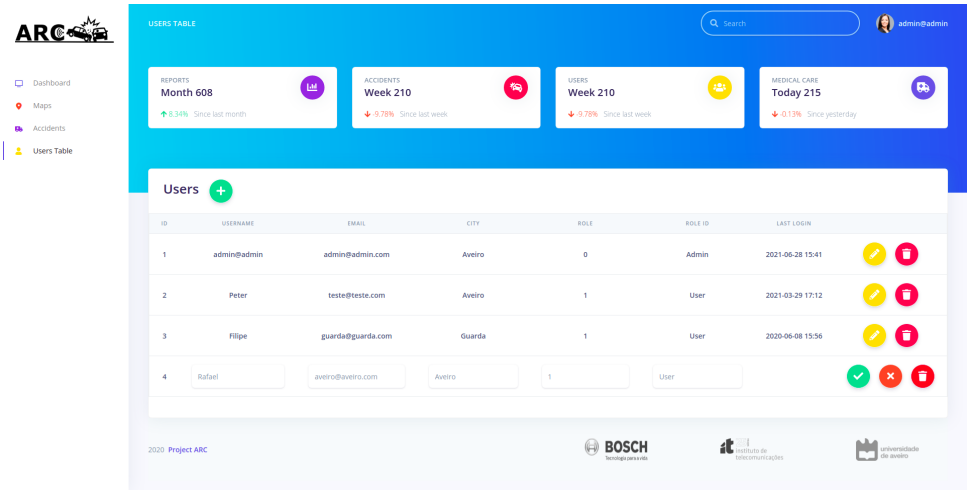

Figure 7-17: Users table in admin view with the option to delete user and edit their data

After the administrator has added a new user to the database, the user can complete his registration on the registration page. The web application only allows users to register if an e-mail address has been previously assigned to them by the administrator and their password is null.

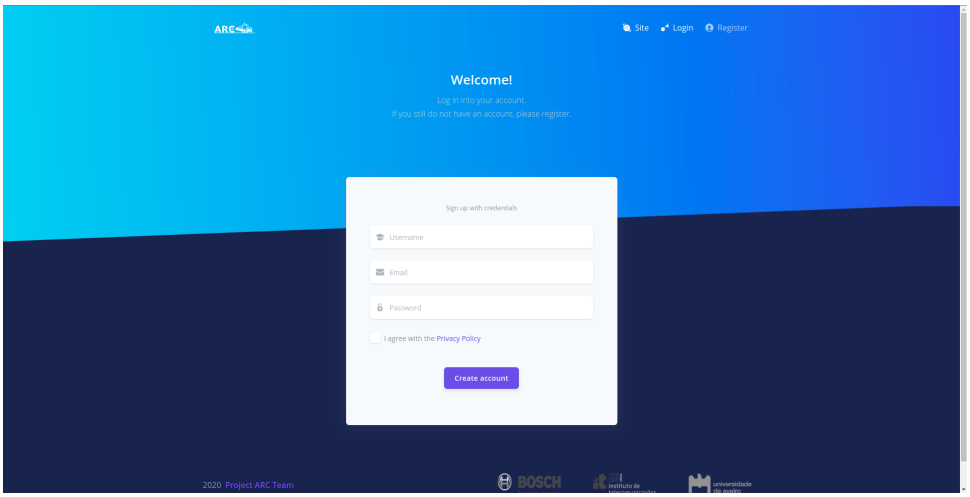

Figure 7-18: User Register page

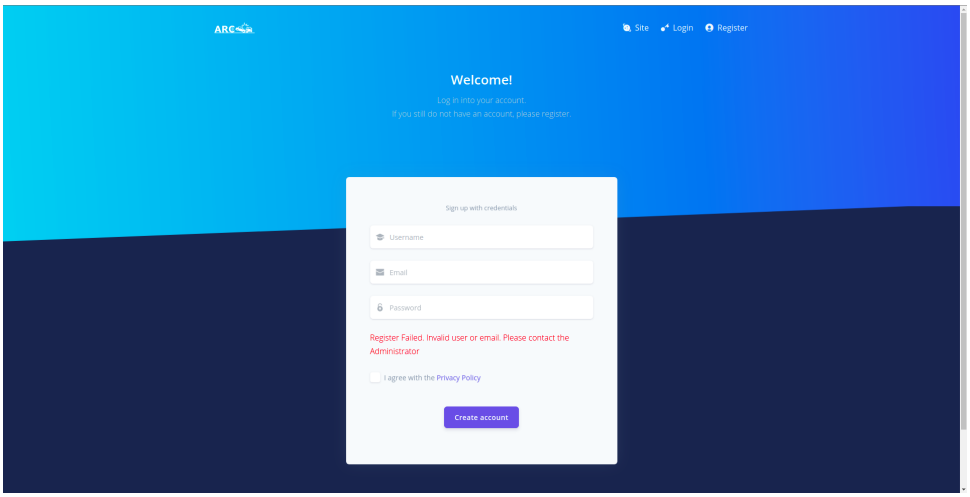

Figure 7-19: User Register page with error

Also, in the accident table, the admin view has its advantages. If the user is an administrator, it is possible to remove an accident from the database by clicking on the trash icon in the accident table.

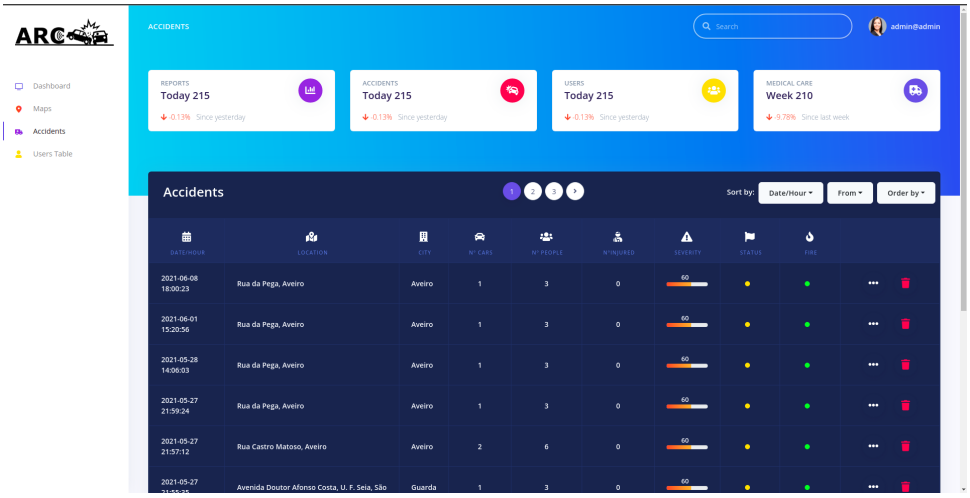

Figure 7-20: Accidents table with option to delete accidents on the web app

8. Conclusion

In this work, the use of vehicular networks and cellular communications has been successfully demonstrated to reduce the response time of emergency services and manage street traffic by autonomously alerting emergency services. This has been achieved by designing software that autonomously detects crashed vehicles, analyses the vehicles surrounding the crash and chooses a gateway vehicle or RSU capable of forwarding an emergency message using the REST API. All the information is processed and displayed in a web application. After the detection of the accident, the emergency message, the cars and cameras near the accident, the remaining accidents, the video and the RTSP link are transmitted in 0.24896, 0.0004258246, 0.01619988 seconds, respectively (on average). After this, users can associate a particular ambulance, seeing then its path and if they wish, choose one of the nearby cars/cameras to see its livestream. The tracking of the position of the cars takes about 0,984 seconds to be registered in the backend of the project (1,11E-05 for the decoding of the position plus 0,9836583138 for the insertion on the InfluxDB, on average) . This work is not intended to replace other existing services, such as eCall, but to work alongside them to provide more information.

Regarding future work, in the web app, the work should focus on creating a dataset for statistics regarding accidents on the front page: how many accidents per month, how many accidents per district and some other filters that could be interesting, add means of communicating with emergency services by implementing text and video chat, improve signposting of congested areas and improve database queries. On vehicular and cellular communication it is important to work on improving the mechanical simulation capabilities of our algorithms such as using SUMO [9] - Urban Mobility Simulation - as a tool to create more test scenarios.

Bibliography

[1] M. S. B. Syed, F. Memon, S. Memon, and R. A. Khan. "iot based emergency vehicle communication system," 2020 international conference on information science and communication technology (icisct), 2020, pp. 1-5, doi: 10.1109/icisct49550.2020.9079940.

[2] The interoperable eu-wide ecall. `https://ec.europa.eu/transport/themes/its/road/action_plan/ecall_en`, 2018.

[3] 5gcar - 5g communication automotive research and innovation. `https://5gcar.eu/`, 2019.

[4] The 5g infrastructure public private partnership. `https://5g-ppp.eu/5gcar/`, 2019.

[5] L. Sequeira, A. Szefer, J. Slome, and T. Mahmoodi. "a lane merge coordination model for a v2x scenario," in 2019 european conference on networks and communications (eucnc), 2019, pp. 198–203.

[6] F. Pereira, H. Sampaio, R. Chaves, R. Correia, M. Luís, S. Sargento, M. Jordão, L. Almeida, C. Senna, A. S. R. Oliveira, and N. Borges Carvalho. "when backscatter communication meets vehicular networks: Boosting crosswalk awareness," ieee access, vol. 8, pp. 34 507–34 521, 2020.

[7] Virginia de Cózar, Javier Poncela, Marina Aguilera, Muhammad Aamir, and Bhawani Shankar Chowdhry. Cooperative vehicle-to-vehicle awareness messages implementation. In Faisal Karim Shaikh, Bhawani Shankar Chowdhry, Habib M. Ammari, Muhammad Aslam Uqaili, and Assadullah Shah, editors, *Wireless Sensor Networks for Developing Countries*, pages 26–37, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[8] José Santa, Fernando Pereniguez-Garcia, Antonio Moragón, and Antonio Skarmeta. Experimental evaluation of cam and denm messaging services in vehicular communications. *Transportation Research Part C: Emerging Technologies*, 46:98–120, 09 2014.

[9] Sumo - simulation of urban mobility. `https://www.eclipse.org/sumo/`.