

DM

Development and Analysis of an Open-source Platform to Simulate Electric Vehicle Charging Needs

MASTER DISSERTATION

Manuel Joaquim Andrade Sousa Perez

MASTER IN INFORMATICS ENGINEERING



UNIVERSIDADE da MADEIRA

A Nossa Universidade

www.uma.pt

September | 2021

Development and Analysis of an Open-source Platform to Simulate Electric Vehicle Charging Needs

MASTER DISSERTATION

Manuel Joaquim Andrade Sousa Perez

MASTER IN INFORMATICS ENGINEERING

ORIENTATION

Filipe Magno Gouveia Quintal

Development and analysis of an open-source platform to simulate electric vehicle charging needs

JOAQUIM PEREZ, Universidade da Madeira

There is the need to improve the charging process of EVs. In order to do that, the field of smart-charging and smart-charging algorithms emerged. Nevertheless, the studies involved in this field are complex, expensive, and risky, leading to a need for prior simulations to analyze/predict the integration of EVs in the electrical networks. There have been some solutions to solve this problem. However, they consist of either academic, proprietary, or limited/rigid solutions. On that account, in this thesis, we have presented a solution that provides a handy and intuitive tool for the researchers to simulate these scenarios with a decoupled and flexible simulation system. Its decoupled architecture is accomplished by adopting open design approaches and the concept of containerized micro-services, easing up the process of maintaining/extending it and providing high scalability. This solution was evaluated in three assessments: migrating it to a remote production system, giving an external developer the task of enhancing a given data model, and integrating this system with an external one. This solution delivered good results in these three tasks. All in all, this solution was motivated by the good aspects of some solutions found in the related work (and improving some of them), it fulfilled its objectives, and it solved the stated problem. At the moment, this solution is already up and running on a production system while also being consumed externally.

Additional Key Words and Phrases: Software Engineering, Energy, Smart-charging, Machine learning, Modelling, Microservices

CONTENTS

Abstract	I
Contents	III
Glossary	VII
List of Figures	IX
List of Tables	XI
1 Introduction	1
1.1 Problem Statement	2
1.2 Proposed Solution	3
1.3 SMILE Project	4
1.4 Structure of the document	5
2 Related Work	7
2.1 Existing simulation tools	7
2.1.1 PerMod	7
2.1.2 SimSES	7
2.1.3 BLAST	8
2.1.4 SAM	9
2.1.5 FreeGreenius	9
2.1.6 PSIM	9
2.1.7 JANUS	10
2.1.8 V-Elph	11
2.1.9 SIMPLEV	12
2.1.10 ADVISOR	14
2.1.11 MARVEL	15
2.2 Data presentation	15
2.3 Simulators' ideal architecture	18
2.4 Conclusions and Solution	19
3 Solution	23
3.1 Elicited requirements	23
3.2 Modelling the solution	23
3.2.1 Travels and battery consumption	25
3.2.2 Charging	26
3.2.3 Affluence	27
3.2.4 Summary and observations	29
3.3 Development tools	30
3.3.1 Core programming language	30
3.3.2 Machine learning	31
3.3.3 Microservices development	32
3.3.4 API development / Static file serving	32
3.3.5 Real-time communication	32
3.3.6 Message brokers	33
3.3.7 Containerization	33

3.3.8	Front-end development	34
3.3.9	Scripting	35
3.3.10	Object-relational mapping	36
3.4	Architecture	36
3.4.1	Simulator	36
3.4.2	Gateway	39
3.4.3	Data models	39
3.4.4	Web client	41
3.5	Functional overview	43
4	Implementation	47
4.1	Git flow	47
4.2	Docker images	47
4.3	Simulator	48
4.3.1	Docker volume	49
4.3.2	Webhook	50
4.3.3	ORM classes	51
4.3.4	REST API	52
4.3.5	WS messaging	52
4.4	Gateway	53
4.5	Data models	55
4.6	Web client	57
4.6.1	Application manifest	57
4.6.2	UI views	59
4.6.3	Configuration	61
4.6.4	REST API	62
4.6.5	WS messaging	62
4.7	Implementation final considerations	63
5	Evaluation and Analysis	65
5.1	Migration of the solution	65
5.2	Enhancement of a data model	65
5.3	Sharing with SMILE partners	67
5.4	Conclusions	67
6	Discussion	69
6.1	Development	69
6.2	Architecture	69
6.3	UI	69
6.4	Deployment and integration	70
6.5	Conclusions	70
7	Conclusion	73
7.1	Future work	74
	References	75

GLOSSARY

AMQP Advanced Message Queuing Protocol. 33, 53

ANL Argonne National Laboratory. 15

API Application Programming Interface. X, XI, 4, 23–26, 30, 32, 36, 43, 52, 53, 61–63, 65, 67

behind-the-meter applications energy applications that provide power that can be used on-site without passing through a meter. 8

BESS battery energy storage systems. 7, 8, 20

BLAST Battery Lifetime Analysis and Simulation Tool. IX, 8, 20, 69, 70

CLI command-line interface. 47, 65, 66

CO₂ carbon dioxide. 1, 2

DLR German Aerospace Center (Deutsches Zentrum für Luft- und Raumfahrt; DLR). 9

EEM *Empresa de Eletricidade da Madeira*. 1

EV electrical vehicle. I, IX, 1, 2, 4, 8, 10–12, 20, 23, 73

Git free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. 8

HEV hybrid electrical vehicle. 10–12, 15, 20

IRR Internal Rate of Return. 9

LCOE Levelized cost of electricity. 9

Li-ion lithium-ion. 8

MVC Model-View-Controller architecture. 34, 41, 57, 59

NPV Net Present Value. 9

NREL National Renewable Energy Laboratory. 8, 9, 14

O&M operations & maintenance. 9

Ofgem Office of Gas and Electricity Markets. 1

ORM Object-relational mapping. 30, 37, 49, 51

PerMod Performance Simulation Model for PV-Battery Systems. IX, 7, 20, 69

PSIM Power electronics simulator. 9, 10, 20, 69, 70

REST Representational state transfer. X, XI, 23, 30, 36, 43, 52, 53, 61–63, 65, 67

RPC Remote Procedure Call. 33, 39, 53, 63, 69

SAM System Advisor Model. IX, 9, 19, 20, 69, 70

SDK software development kit. 9

SIMPLEV simple electric vehicle simulation program. IX, 12–14, 20, 70

SimSES simulation of stationary energy storage systems. IX, 7, 8, 19, 20, 69, 70

SMILE Smart Islands Energy System. IX, XI, 4, 5, 18, 23, 24, 65, 67, 70, 73

stationary applications energy applications that are attached to a fixed site, being land, a building or other immobile structure for extended use at that site, and includes stationary electrical power systems which are transportable for temporary site power supply. 8

UI user interface. 16, 57–60, 63, 70, 71

UX user experience. 34

WS WebSocket. X, XI, 30, 32, 36, 41, 52, 53, 61–63, 65, 81, 82

WSGI Web Service Gateway Interface. 32

LIST OF FIGURES

1	EV usage - statistics	1
2	Functional diagram	3
3	PerMod - schematic structure	7
4	SimSES - modelling overview	8
5	BLAST - overview	8
6	SAM - overview	9
7	FreeGreenius - showcase	10
8	PSIM - overview	10
9	JANUS - overview	11
10	V-Elph - overview	12
11	SIMPLEV - initial screen	12
12	SIMPLEV - input prompts	13
13	SIMPLEV - simulation output	13
14	SIMPLEV - simulation output - graphics	14
15	ADVISOR - overview	14
16	MARVEL - input data and configuration	15
17	MARVEL - simulation results	16
18	Chevin	17
19	Fleetio	17
20	GFI Systems	17
21	Johns Hopkins COVID-19 Dashboard	17
22	SMILE Tukxi Dashboard	18
23	Travel route example	18
24	Monolithic approach vs. Microservice approach	19
25	Travel distance sample	26
26	Travel battery consumption sample	26
27	Charging period duration sample	27
28	Charging peak value sample	28
29	Affluence - results' representation	29
30	Python - snippet	31
31	TensorFlow - snippet	31
32	Nameko - snippet	32
33	Flask - snippet	32

34	websockets - snippet	33
35	RabbitMQ - snippet	33
36	Docker - snippet	34
37	OpenUI5	34
38	Chart.js	35
39	Makefile	35
40	SQLObject - snippet	36
41	Simulator - ER Diagram	37
42	Core component - architecture	38
43	Gateway component - architecture	39
44	Travel affluence model - architecture	39
45	Travel distance model - architecture	40
46	Travel duration model - architecture	40
47	Travel battery consumption model - architecture	40
48	Charging period duration model - architecture	40
49	Charging period energy consumption model - architecture	40
50	Client component - architecture	42
51	Use cases	43
52	Simulation flowchart	44
53	Travel flowchart	45
54	Charging period flowchart	46
55	Git flow	47
56	Build process example	47
57	Run process examples	48
58	SQLite database	50
59	Slack Webhook messages	51
60	i18n example	60
61	Event handling example	61
62	REST API usage	62
63	Example of WebSocket messages	63
64	Enhancement of a data model	66
65	Docker containers	77

LIST OF TABLES

1	List of contributions	4
2	Related work - Summary	20
3	Elicited requirements	23
4	SMILE Tukxi API endpoints	24
5	Travels - results	25
6	Charging - results	27
7	Affluence - results	28
8	Data models - Summary	29
9	Simulator configuration	50
10	REST API endpoints	53
11	Gateway endpoints	55
12	Web client configuration	61
13	Types of WebSocket messages sent	81
14	Types of WebSocket messages received	82

1 INTRODUCTION

Global warming is one of the biggest challenges mankind is currently facing. To address this issue, there has been a wide variety of measures taken around the globe involving a broad set of areas, namely public campaigns, and greener technologies. This thesis will focus on work from two of the most critical areas - energy and transportation.

One of the most prominent factors affecting global warming is the constant carbon dioxide (CO₂) emissions by the majority of the vehicles used in our day-to-day routines.

An approach taken to reduce these emissions is to shift the transportation sector towards less polluting alternatives. One of these alternatives is the adoption of electrical vehicles (EVs)[28] (its worldwide adhesion is illustrated in Figure 1).

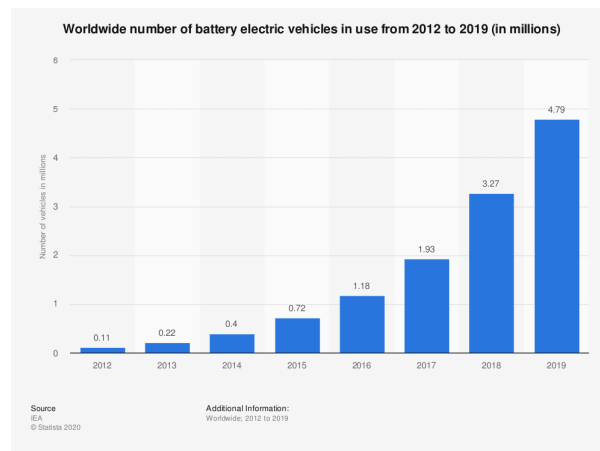


Fig. 1. EV usage - statistics

This shift promoted the ascension of EVs currently produced and offered by manufacturers. For the same reasons, governments around the globe are also actively encouraging the adoption of these vehicles through, for example, tax breaks or free parking. For example, in the case of Madeira Island, according to a report from *Empresa de Eletricidade da Madeira* (EEM)[1], the quantitative adhesion in Madeira Island is growing.

On the other hand, as much as the electrical networks have grown through the years[40], they still end up behind the EVs' evolution[18] and not able to sustain the growing demand for EVs. That is, the increased usage of EVs leads up to a bigger demand in the electrical networks, and these networks are considered outdated and not ready for this new demand[37]. For example, according to a Ofgem report, it is estimated that about a third of the electrical networks across the UK would need to be upgraded if approximately 40% of its customers shifted to EVs[32].

Consequently, it is expected that we will recur back to fossil fuels to produce enough energy to address this increasing demand[25]. This obstruction takes us back to our original problem presented in the first place (our CO₂ footprint).

In other words, there is a necessity to effectively charge these EVs without overloading the electrical networks while still being able to respond to its demand and leave fewer human footprint on our ecosystem.

Traditionally, the charging process involves the *plug-and-charge* charging method, which consists of plainly charging the vehicle in an uncontrolled/uncoordinated way whenever it needs to be charged. This paradigm causes the theoretical issues involving the electric networks' inability to address the demand and be able to charge these EVs without overloading the network, as described in the paragraph above. To address the issue mentioned previously, the field of Smart-Charging[49] emerged, bringing up intelligent and coordinated charging methods that take into account several factors such as energy cost and electricity network availability. Recurring to a diverse set of algorithms, these coordinated charging methods lead up to a more balanced distribution of the load on the electrical networks and lessened peak values[26][32]. As a result, these methods end up managing the electrical networks' loads in such a way that we can then safely and assuredly adhere to EVs (and effectively minimize our CO₂ footprint).

1.1 Problem Statement

From the practical standpoint, the studies involving smart-charging algorithms are feasible yet difficult, expensive, and dangerous, since it requires complex management and coordination in terms of the electric power and road transport systems[23]. Thus, to successfully analyze/predict the integration of EVs in the electrical networks and also (very importantly) for the grids' safety, these studies are usually firstly conducted recurring to simulations[23].

From the data standpoint, the lack of data is also an issue when it comes to evaluating algorithms and machine learning in general[15], being that simulations will also help in augmenting/enlarging the datasets and improving decision-making/prediction in this kind of systems[4].

Not to mention the additional impact of the Covid-19 pandemic has had on the lack of data, particularly in areas such as energy and transportation, since there are not many drivers, cars, and travels to evaluate proposed algorithms. Therefore, as usual in the area of smart-charging (and because of the reasons mentioned previously), the lack of data (or the lack of non-noisy data) leads up to the testing being done based on simulation of factors such as the usage of EVs, the grid's load, the cost of energy and the charging of batteries in general [27][43][16][49].

There are simulation solutions available that fit different scenarios. Still, they consist of either academic, proprietary solutions, or limited/specific solutions to a particular context that require a bit of effort from the researchers/practitioners to customize them to their intended context (even if the required changes are minimal)[19][14]. Furthermore, the literature review carried out in this thesis disclosed that, in terms of architecture, the existing solutions are usually made in a monolithic way, containing a single code base and leading up to more rigid solutions and challenging to adjust/customize as intended by the end-user. Besides that, technologically speaking, there are not many available frameworks that integrate this kind of simulation while being open-source, extensible, and easy-to-use[19][14], which obstructs its reusability or customization in other contexts.

To overcome this, we intend to develop a simulation platform that allows researchers to simulate any set of smart-charging algorithms in different conditions. In other words, the main goal of this thesis consists of building such a solution with enough abstraction to fit any charging simulation context.

1.2 Proposed Solution

The solution will consist of an open-source system that simulates smart-charging algorithms, developed in an abstract way that will consider any context and a variable set of data models. Simply put, it will be developed modularly both in terms of simulation algorithms (smart-charging algorithms) and in terms of simulation components (described by a user-defined set of data models - e.g., battery, charger, charging time).

Besides that, this simulation system will contain three other components - a Data Server (to expose the simulation data in a Web Server), a Web Client (to provide an interactive dashboard regarding this simulation system), and a Slack Webhook (to send notifications to a Slack channel regarding the start/stoppage of simulations and possible errors/exceptions that may occur in the system).

It will also consist of a system that will be technically open and easily extensible to any energy simulation context. Its functional representation can be observed at Figure 2.

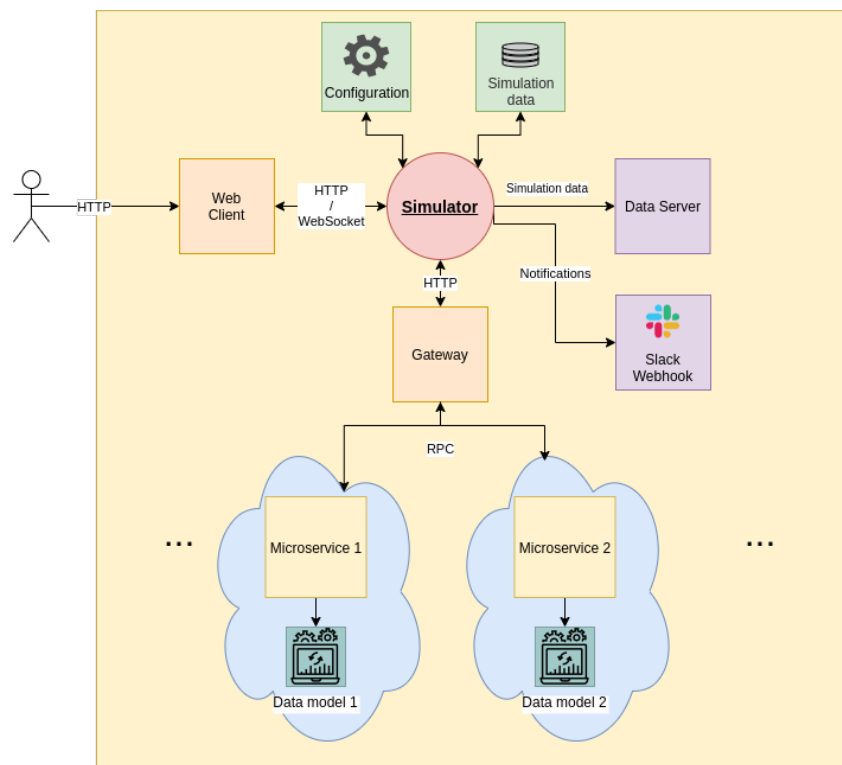


Fig. 2. Functional diagram

This solution and its architecture will be designed to consider a case study (mentioned later in the document). Still, it will be built abstractly and openly enough to fit any other kinds of contexts and areas.

In terms of this thesis' contributions, these are listed in Table 1.

Contribution	Description
C1	State of the Art revision considering platforms for simulations
C2	Proposal of an architecture for the solution
C3	Construction and implementation of the system

Table 1. List of contributions

1.3 SMILE Project

The solution presented in this thesis will include a case study involving the project known as Smart Islands Energy System (SMILE).

This project is funded by the European Union's "Horizon 2020 research and innovation programme", which consists of a demonstration of nine different smart grid technologies on three different islands[42]. The end goal of the project is to foster the market introduction of these nine technologies[42]:

- Integration of battery technology
- Power to heat
- Power to fuel
- Pumped hydro
- EVs
- Electricity stored on board of boats
- Aggregator approach
- Demand side management
- Predictive algorithms

It features the collaboration of nineteen partners from various European countries and includes companies, research institutions, regional governments, and cluster organisations[42]. In terms of its duration, it started on 1 May 2017, and it was scheduled to end on 30 April 2021. This project has been the source of many scientific articles, such as [2][21][22].

As previously mentioned, the field of smart-charging emerged from the necessity of making the electrical networks capable of sustaining the increased adhesion of EVs. Regarding the context of the SMILE project in Madeira Island, a EV pilot came up to implement smart-charging techniques in vehicles of small size and small energy requirements (e.g., scooters). As a result, a hardware/software solution was developed that utilizes commercial equipment and low-cost sensors, allowing to cut and provide energy at any charging point and provide an interface of visualization and control of the system. This pilot also depended on the crucial collaboration of its drivers, who indicate the battery's state on a scale from 0 to 10 (being that 0 equals 0% and 10 equals 100%) and the traveled distance between charging periods through a mobile application. The functionality of this solution is available through an Application Programming Interface (API) that allows the charging algorithms to be implemented separately from the control, granting the test of several algorithms with zero-cost integration. During this project, this system was installed in a Madeiran company that offers a city tour around Funchal and its surroundings through scooters, both electric-fueled and gasoline-fueled. These scooters are known as Tukxis.

However, due to the Covid-19 pandemic, there is a lack of drivers and infrastructures to test the solution mentioned above, making the project very adequate for the solution presented in this thesis.

Therefore, this project's data will be used to model the tool's test case. In other terms, SMILE data regarding Madeira Tukxi drivers' will be gathered and modeled to build a test case for this thesis.

1.4 Structure of the document

This document is structured in the following chapters: **Related Work, Solution, Implementation, Evaluation and Analysis, Discussion and Conclusions.**

The first section's content will consist of a literature review and research targeting the subject addressed in this thesis, alongside the gathered conclusions and points taken in the perspective of the solution built on this thesis.

Then, the next section will describe the case study of this thesis (and its modelling), the solution's high-level architecture and requirements elicitation, together with its representation in diagrams (such as a class diagram, event diagram), as well as its functional description and development tools.

In the same way, this thesis will include a section that will describe the solution's implementation process (e.g., development packages, the configuration used, snippets of relevant scripts/code) and the technical approaches used.

The above section will be followed by another containing the results of the developed solution when considering the case study provided by the SMILE project. This section will be able to answer questions, namely "For how much time can we run the solution?" and "How easily can we add/remove a certain module?".

The discussion chapter will discuss the results considering the literature review and the proposal of the solution. Furthermore, the decisions made during its implementation will also be analyzed and discussed regarding whether they were fulfilled or not and whether they were well-executed or not.

Lastly, the final section will be composed of the summary of the whole thesis, alongside indicating the learned lessons and the future work associated with the thesis itself.

2 RELATED WORK

As previously mentioned, some solutions are already targeted at the problem we identified for this work. Thus, in this section, while fulfilling contribution C1, those solutions will be studied to gather motivation for our approach. For each tool, we will briefly present its contribution. Their respective pros and cons will also be analyzed in order to fulfill the idealization of this thesis' solution.

2.1 Existing simulation tools

Since this thesis will involve a simulation tool, existing simulation tools will be presented and discussed in this subsection.

2.1.1 PerMod

When it comes to battery energy storage systems (BESS), the necessity of reducing energy costs and increasing its efficiency is always of the most importance.

Since there is a panoply of adjustable details related to a storage system design, the University of Applied Sciences of Berlin developed a system in MATLAB - Performance Simulation Model for PV-Battery Systems (PerMod)[47] - that enabled the end-user to simulate and analyze the performance of a particular energy storage system. It considers several input parameters regarding power losses (e.g., conversion losses, standby losses) processed in the simulation model. Its schematic structure is shown in Figure 3.

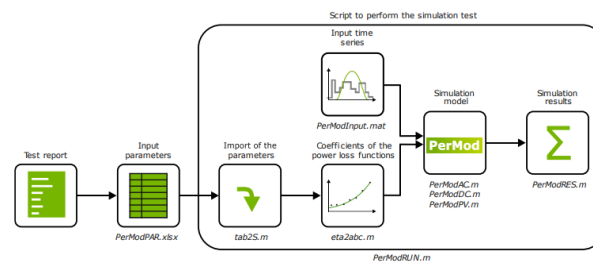


Fig. 3. PerMod - schematic structure

As a limitation, in this tool, the BESS degradation is being neglected. Furthermore, given that this tool was written in MATLAB, it results in a much inflexible and platform-rigid solution.

2.1.2 SimSES

Also in the context of evaluating BESSs, a software system with the capability of performing a simulation of stationary energy storage systems (SimSES)[30] was developed.

This simulation system allowed the analysis not only from the technical but also from the economic standpoint. In other words: from the technical perspective, it provides the capability of analyzing the efficiency and the impact of specific control algorithms; from the economic point of view, it enables the researcher to compare a set of different electronic components to optimize the device's profitability. The overview of its model can be observed in Figure 4.

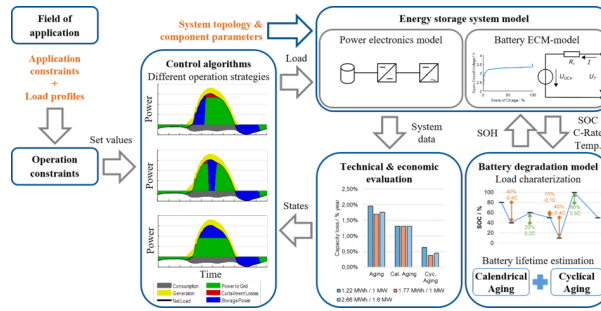


Fig. 4. SimSES - modelling overview

In its initial phase, this tool was developed in MATLAB. Eventually, it ended up being ported to an open-source Python software[41] (leading up to a more technically modern and flexible solution). Its Git repository can be consulted at <https://gitlab.lrz.de/open-ees-ses/simses>.

2.1.3 BLAST

Focusing in the precise context of BESSs and electric vehicles, and with the goal of economically evaluating EVs, stationary applications and behind-the-meter applications, the National Renewable Energy Laboratory (NREL) developed the Battery Lifetime Analysis and Simulation Tool (BLAST) system[31] - so as to predict battery responses, according to battery properties (such as its degradation and its thermal performance), its usage (e.g. driving data) and historic climate data.

BLAST’s model and an example representation of its simulation results are shown in Figure 5.

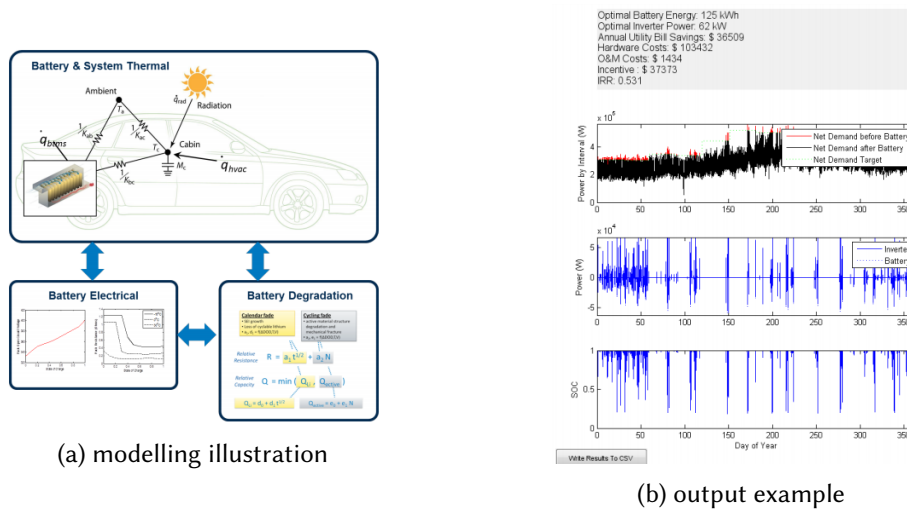


Fig. 5. BLAST - overview

As a result, this tool eases up the process of optimization and deployment of Li-ion batteries.

Unfortunately, it is pretty restricted programming-wise since the researcher only has access to its installer (and its binary files).

2.1.4 SAM

Under the more specific context of renewable energy, the National Renewable Energy Laboratory (NREL) developed and distributed a simulation system called System Advisor Model (SAM)[5]. It consists of a desktop application that allows the end-user to simulate renewable energy projects and examine techno-economical factors, such as performance, financial metrics, and incentive options for that kind of project.

The platform is implemented as an open-source platform built on top of C/C++, which brings up a few pros and cons. On the positive side, it leads up to a more optimized and portable platform. However, it also results in a more low-level development (and therefore, involving more rework and less reusability).

Nonetheless, besides it being open-source (thus easily allowing custom enhancements on it), its versatile software development kit (SDK) enables the creation of additional simulation modules under a set of possible programming languages that includes C/C++, C#, Java, Python, and MATLAB. Overall, the usage of SAM can be described in Figure 6.



Fig. 6. SAM - overview

2.1.5 FreeGreenius

Within the same field of research (renewable energy), which is characterized by the unpredictability of its sources (such as solar and wind), the German Aerospace Center (Deutsches Zentrum für Luft- und Raumfahrt; DLR) (DLR) developed FreeGreenius[11]. Based on the given input (meteorological and economic data), the tool is capable of simulating, from the economic point of view, renewable power plants, taking into account investment costs, operations & maintenance (O&M) costs and financing costs, considering a certain period. As a result, it displays a set of economic metrics, such as payback times, IRR, LCOE, and NPV. This tool is illustrated in the pictures of Figure 7.

2.1.6 PSIM

In order to simulate power electronic converter and motor drives in general, the proprietary software known as Power electronics simulator (PSIM)¹ was developed. This tool allows the development/modelling of custom module boxes.

¹<https://powersimtech.com/products/psim>

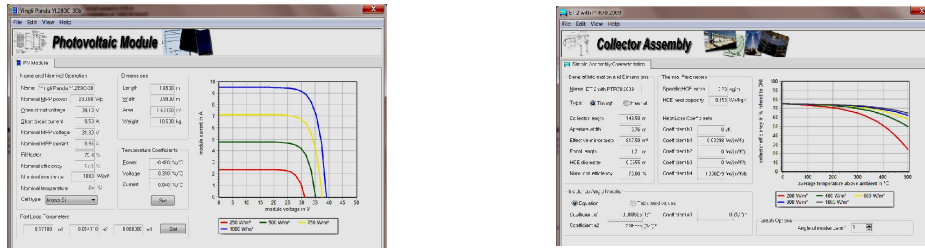


Fig. 7. FreeGreenius - showcase

In 2004, the development of custom module boxes targeting the simulation of automotive systems arose[33]. As a result, these custom developments enabled this tool to simulate and study conventional vehicles concretely, EVs and HEVs.

As similarly seen in the simulators above, this system outputs its simulation results graphically. An example of a PSIM model and its output is shown in Figure 8.

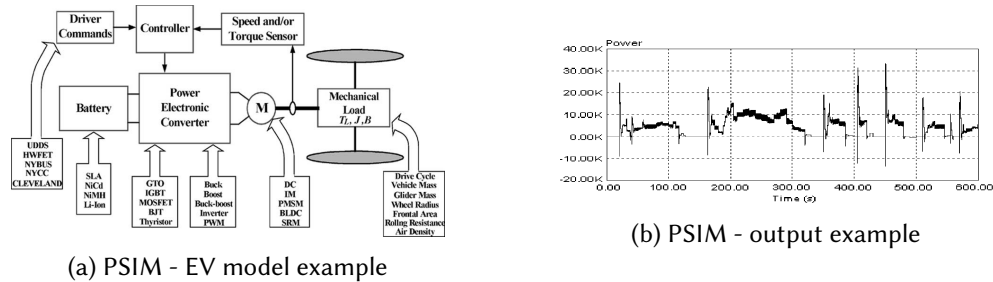


Fig. 8. PSIM - overview

2.1.7 JANUS

Further on the same context (automotive systems), the JANUS[6] simulation package was developed in the Engineering Department at Durham University. It consists of a program that allows the end-user to evaluate the design, performance, and (energy) efficiency of vehicles (traditional, battery-electric, or hybrid-electric ones).

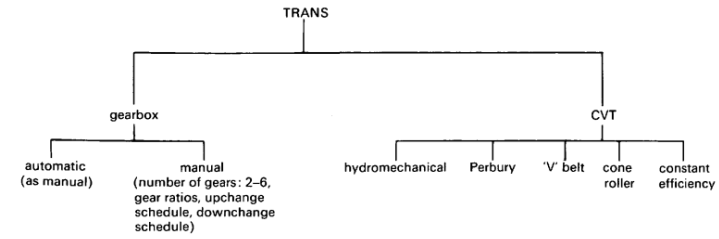
Fortunately, during its development, the program was written so that it allowed possible extensions. In order to do that, its structural approach consisted of separating each vehicle component in a separate subroutine.

These subroutines are also subdivided into three sections - the initial section (responsible for handling parameters and information about the vehicle itself), the dynamic section (the central computational part of the program), and the output section (that displays simulation details such as the vehicle, its components (and efficiencies/losses) and its driving cycle).

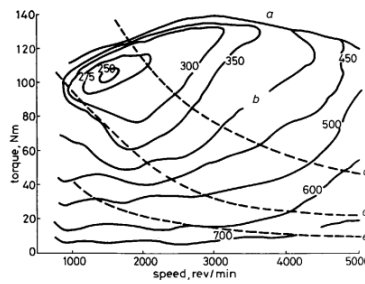
In Figure 9 we can observe a bit of its structure and an example of its output.

Component	Simulation name
Vehicle definition	VEHICLE
Driving cycle	DCYCLE
Wheels	WHEELS
Final drive	AXLE
Transmission	TRANS
Clutch or torque convertor	COUPLE
Internal combustion engine	ICENG
Series DC motor	DCSER
Separately excited DC motor	DCSHUNT
DC switched reluctance motor	DCREL
AC induction motor	ACINDUC
DC generator	DCGEN
Field chopper	FCHOPR
Armature chopper	ACHOPR
Traction battery	BATTERY
'Gearing' for connecting two prime movers	DRIVE
Battery switching	BATSWCH
Summing block	SUM
Torque splitting module (hybrids)	TORQSPLT
Power splitting module (hybrids)	POWSPLT
Vehicle controllers	VEHCONT

(a) list of simulation routines



(b) TRANS routine - subdivision



a Maximum torque curve
 b Constant SFC 400g/kWh
 c Power for constant 115 km/h
 d Power for constant 80 km/h
 e Power for constant 50 km/h

(c) output example

Fig. 9. JANUS - overview

Unfortunately, JANUS was developed in FORTRAN, an outdated programming language. On one side, FORTRAN is still relatively fast and frequently used for pure mathematical calculations. However, on the other side, regarding factors such as connectivity and support in general, FORTRAN is considered obsolete. Therefore, the programming language used ended up being a shortcoming of this tool.

2.1.8 V-Elph

Succeeding the subject of automotive systems, the Texas A&M University developed a MatLab/Simulink simulation/modeling package - V-Elph[7] - to ease the analysis and comparison of EV and HEV setups and energy management strategies.

This tool allows the end-user to perform simulations based on the selected component model (including component models included for general use-cases and user-defined/customized ones).

In addition, after simulating a specific component model, this package generates a graphical representation of its results.

Then again, since this tool is written in MATLAB, it can be concluded that it is a much limited and rigid solution. Figure 10 illustrates a model example and also showcases its selection screen and an output example.

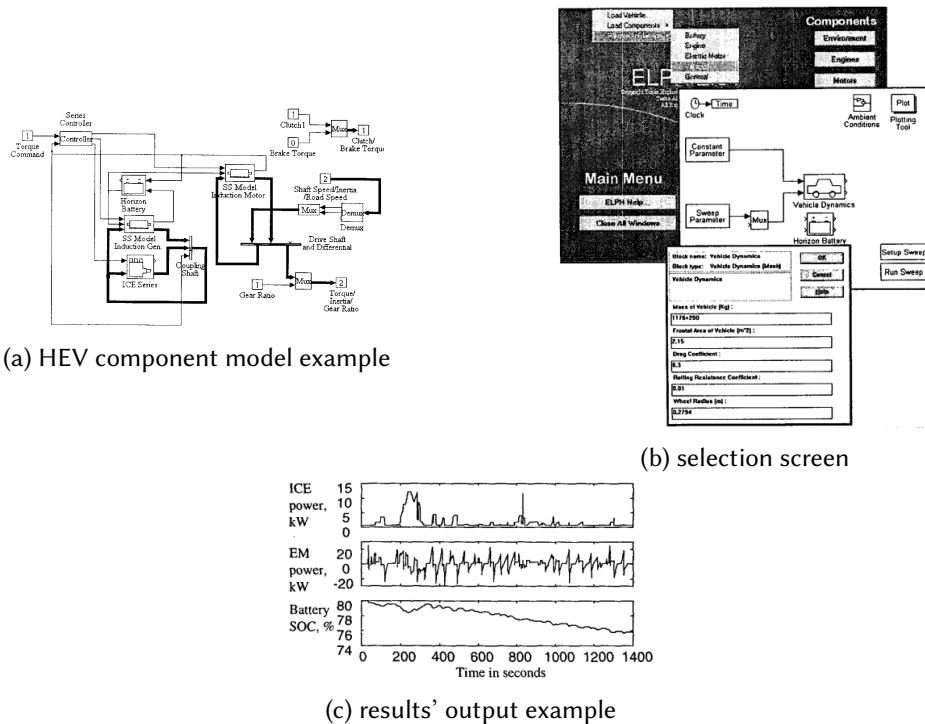


Fig. 10. V-Elph - overview

2.1.9 SIMPLEV

G. H. Cole proposed a simple electric vehicle simulation program (SIMPLEV)[10] as an alternative to systems targeting vehicles' simulations. It consisted of a program written in BASIC that provided the end-user the possibility of performing parametric studies on EVs.

The execution of the program leads up to the rendering of an initial screen (visible in Figure 11), with its main options.

```

*****
*                               *
*           SIMPLEV             *
*   A Simple Electric Vehicle Simulation   *
*           Version 1.0   May 1991   *
*                               *
*   Idaho National Engineering Laboratory *
*   Electric & Hybrid Vehicle Program  *
*           EG&G Idaho, Inc.         *
*                               *
*****

Work supported by the U. S. Department of Energy
Assistant Secretary for Office of Propulsion
Systems under DOE Contract No. DE-AC07-76ID01570.

'D' or 'G' runs IDSEP test case.
<F1> - accesses DOS
<F7> - toggles graphs
<F8> - toggles program execution speed
<F9> - ends this program without printing results
<F10> - ends this program and prints results
Hit space bar to continue....
    
```

Fig. 11. SIMPLEV - initial screen

By advancing to the simulation, a set of input prompts are displayed (that can be observed at Figure 12) to introduce any necessary data for the intended simulation.

```

VEHICLE MENU
Enter number of vehicle for initial parameters:
1 - IDSEP
2 - ETX-I (Ford)
3 - ETX-II (Ford)
4 - ETV-I (Chrysler)
5 - TB-1 (Eaton DSEP)
6 - GM Impact (approximation)
7 - TEVan (Chrysler)
8 - EVcort (Soleq)
9 - Soleq Van
0 - Bedford Van
A - TB-2 (Eaton DSEP)
B - Modular EV - 75hp (Ford)
G - G-Van
U - User written vehicle file
S - DOS access

MOTOR MENU
1 - IDSEP (AC)
2 - ETX-I (AC induction, 35 hp)
3 - ETX-II (AC IM, 70 hp)
4 - ETV-I (DC, separately excited, 20 hp, includes controller)
5 - DSEP TB-1 (AC induction, 60 hp)
A - Modular EV (50 Hp)
B - Modular EV (75 Hp)
C - Modular EV (100 Hp)
U - User written motor file
S - DOS access

Enter number/letter for motor:

INVERTER/CONTROLLER MENU
1 - IDSEP
2 - ETX-I (AC transistor)
3 - ETX-II (AC transistor)
4 - ETV-I (DC, data included in motor file)
5 - TB-1 (AC transistor)
A - Modular EV (50 Hp)
B - Modular EV (75 Hp)
C - Modular EV (100 Hp)
U - User written inverter/controller file
S - DOS access

Enter number/letter of inverter/controller:
    
```

Fig. 12. SIMPLEV - input prompts

As a result, the program prints out the simulation results in three paper sheet pages (including the input data and all calculated results). An example of its printed pages is displayed in Figure 13.

```

Date: 07-16-1991    TIME: 13:08:16    Page 1
*****
1 - SIMPLER - SOURCE AND SIMULATION PROGRAM - VEHICLE I.C. - 4
(LANS National Engineering Laboratory)
*****
Calculated data written to SMOE.DAT for 1 cycles (1 150 - 300) sec
Other data written to SMOE.DAT for 1 cycles (1 150 - 300) seconds
VEHICLE CHARACTERISTICS BASED UPON (DSEP FROM DOE/DOE/10/10/10)

Vehicle Characteristics:
Vehicle test weight: 1048 lbs., 2297 kg
Battery weight: 1200 lbs., 545 kg
Total weight: 2248 lbs., 1012 kg
Frontal area: 22.00 sq. ft., 2.56 sq. m.
Geardown coefficient, G1: 0.0089
Geardown coefficient, G2: 1.0000
Geardown coefficient, G3: 1.0000
Geardown coefficient, G4: 1.0000
Time rolling radius: 11.22 in., 0.29 m
Wheel bearing drag: 0.00 lbs-ft, 0.00 m-hk
Accessory load: 300.00 watts
Deep discharge power fraction: 1.00

MINIMUM COMPONENT INFORMATION:
IDSEP - Soleq Inverter (DSEP, ICM data file)
(Scaled Speed x 1.00, Torque x 1.00)
Maximum power/contoller voltage: 120.0 volts
Maximum inverter/controller current: 422 amperes

DSEP DriveTrain (DSEP, NOT data file)
(Scaled Speed x 1.00, Torque x 1.00)
Scaled motor characteristics are on page 3.
DSEP - Soleq Transaxle (DSEP, TX data file)
(Scaled Speed x 1.00, Torque x 1.00)
First gear ratio: 15.50
Second gear ratio: 10.15
Speed at gear change: 25.7 mph ( 22.2) mph

BATTERIES (NF170-BT data file)
Battery Name: ..... NF170
(Scaled Amp x 100 amp = 100 (C/20) amp 24.0 = 0.1370)
Rated ampere-hour capacity (C/2 Rate): ..... 130.0 Ah
Number of modules: ..... 30
DOD at start of driving cycle: ..... 0.7
Fractional energy capacity remaining @ 75.0 mph: ..... 12.8
Effective capacity at beginning of driving cycle: ..... 180.0 Ah
Average power during pre-discharge: ..... 12.8 hp
Efficiency during pre-discharge: ..... 96.8 %
AUXILIARY POWER UNIT (APU):
On at 80.0% battery DOD
Off at 50.0% battery DOD
Run ended as requested after 1 cycles.

Initial simulation time: 07-16-1991    TIME: 13:10:16    Page 2
*****
Calculated component times (1 1.00 cycle)
60 - 10 mph = 77.7 sec
20 - 10 mph = 19.7 sec
55 - 45 mph = 11.8 sec

Constant speed powertrain loads:
29.7 kW @ 40 mph, 24.6 kW @ 35 mph, 20.2 kW @ 30 mph, 16.3 kW @ 25 mph
11.3 kW @ 40 mph, 10.6 kW @ 35 mph, 9.3 kW @ 30 mph, 8.4 kW @ 25 mph
4.7 kW @ 20 mph, 3.3 kW @ 15 mph, 2.1 kW @ 10 mph, 1.0 kW @ 5 mph

RESULTS OF FUDS SIMULATION (FUDS, C1C, 81 = 1.0 sec.)
Avg density: 0.00216 slugs/ft^3 (0.0329 lb/ft^3)
Road grade: 1.0 %
Vehicle heading: 30 degrees from North
Wind: 10.0 mph from 15 deg, from North
Wind corrections: Cd = 1.33 * (1+0.0004*tau*(1.6370)), tau = 17.5 deg.
= 0.456, tau = 17.5 deg.

Maximum battery power: ..... 65.7 kW (154 hp, 428 hp)
Average battery current: ..... 79.2 disch., 35.2 chg., 61.0 net
Average battery power, kW: ..... 12.8 disch., 10.7 chg., 10.456 net
Rader-hours rained: ..... 28.1 hr
Effective battery capacity: ..... 131.1 Ah
Net battery energy: ..... 3.785 kWh
Gross battery energy: ..... 4.326 kWh
Energy supplied by APU: ..... 0.000 kWh
Energy supplied by regen: ..... 0.377 kWh
Percent of energy supplied by regen: ..... 12.7 %
Percent of energy supplied by APU: ..... 0.0 %

Detailed Results:
Average battery efficiency: ..... 92.7% disch., 96.9% chg.
Average inverter efficiency: ..... 100.0% driving, 100.0% regen
Average motor efficiency: ..... 70.3% driving, 71.1% regen
Average transmission efficiency: ..... 100.0% driving, 98.20% regen
Average powertrain efficiency: ..... 70.3% driving, 62.0% regen

Net battery energy economy: ..... 324.9 wh/mph, 332.3 wh/mph
Gross battery energy economy: ..... 408.9 wh/mph, 378.4 wh/mph

Maximum battery power density: ..... 242.6 w/ft^3, 110.3 w/kg
Average speed: ..... 17.5 mph, 31.4 km/h
Total distance traveled: ..... 7.4 mi, 12.0 km
Vehicle driving time: ..... 0.383 hours ( 1372 seconds)
Number of cycles completed: ..... 1
DOD at termination: ..... 22.0 %
Battery voltage at termination: ..... 182.4 volts
Battery current at termination: ..... 2.7 amperes

AVERAGE COMPONENT POWER LOSSES FOR THIS RUN
Average Power Losses:
Battery: 0.931 kW disch., 0.044 kW chg., 0.975 kW total
Inverter: 0.000 kW driving, 0.000 kW regen., 0.000 kW total
Motor: 3.283 kW driving, 0.813 kW regen., 4.096 kW total
Transmission: 0.000 kW driving, 0.282 kW regen., 0.282 kW total

STEADY STATE POWERTRAIN EFFICIENCIES UNDER ABOVE ROAD CONDITIONS:
0.728 @ 40 mph, 0.722 @ 35 mph, 0.706 @ 30 mph, 0.689 @ 25 mph
0.467 @ 40 mph, 0.468 @ 35 mph, 0.417 @ 30 mph, 0.386 @ 25 mph
0.211 @ 20 mph, 0.349 @ 15 mph, 0.470 @ 10 mph, 0.386 @ 5 mph

SIMULATION DIAGNOSTICS:
Maximum motor power occurrences: 3
Extrapolated motor efficiency (torque): 0 times
Extrapolated motor efficiency (speed): 0 times
Extrapolated transmission efficiency (torque): 0 times
Extrapolated transmission efficiency (speed): 0 times
Extrapolated inverter efficiency (torque): 0 times
Extrapolated inverter efficiency (speed): 0 times

Scaled motor characteristics:
Motor Speed Max. Torque Peak Power
(rpm) (lb-ft) (hp)
Motor
0 86.0 0.0
500 86.0 6.0
1000 86.0 14.0
1500 86.0 24.0
2000 86.0 36.0
2500 86.0 48.0
3000 86.0 60.0
4000 60.0 68.5
5000 75.0 68.5
6000 60.0 68.5
7000 48.0 68.0
8000 48.0 73.1
9000 30.0 61.7
10000 24.0 43.7
    
```

Fig. 13. SIMPLEV - simulation output

Additionally, SIMPLEV also outputs its simulation results in a graphical format (being that it could be shown on display or printed on paper). This output format is exemplified in Figure 14.

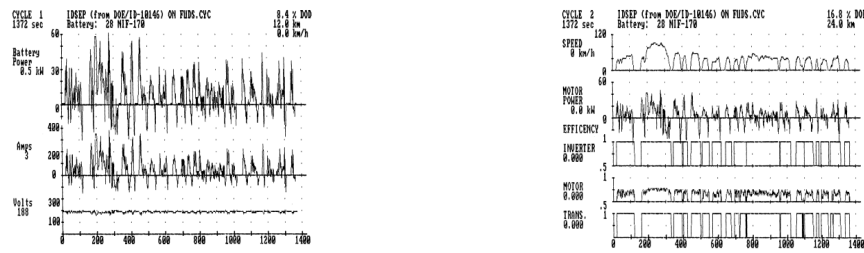


Fig. 14. SIMPLEX - simulation output - graphics

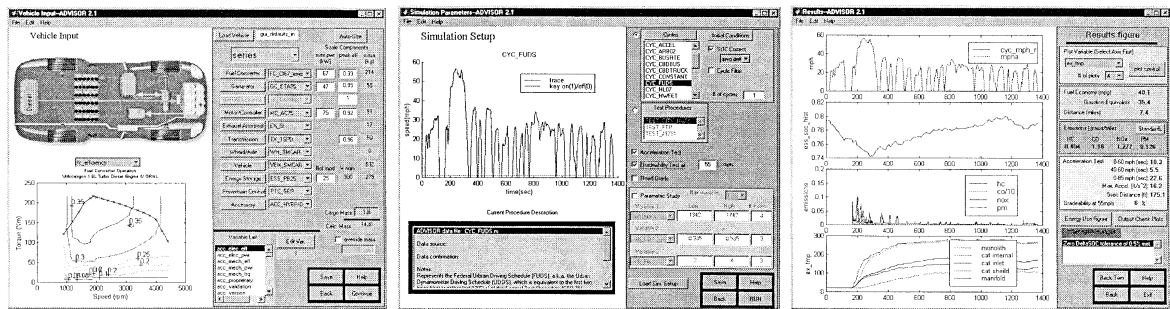
Considering it was developed in BASIC, SIMPLEX turns out to be a significantly restricted and bounded solution due to the minimal capabilities of the programming language used to build it.

2.1.10 ADVISOR

Another tool with similar objectives, known as ADVISOR[48], emerged from NREL. The previous simulators came up short when comparing to this one in terms of availability of its code (this one was made publicly available, encouraging continuous development) and also in terms of flexibility (since the developed modules sometimes did not fully represent certain scenarios, this simulator enabled modifications/enhancements).

In a nutshell, in its first step, it allows the end-user to select a particular vehicle and to parameterize its configuration. Accordingly, the end-user also has the capability of configuring the whole simulation procedure. At last, after the simulation has run, ADVISOR outputs its results.

Figure 15 is used to showcase these three steps (selecting a vehicle, setting up the simulation, and observing its results).



(a) vehicle input screen

(b) simulation setup screen

(c) results screen - example

Fig. 15. ADVISOR - overview

As ADVISOR was built on MATLAB, its portability and versatility are hindered compared with other work reviewed in this section.

2.1.11 MARVEL

Likewise, in order to analyze HEV systems, at Argonne National Laboratory (ANL) the MARVEL[24] program was developed.

For the simulation itself, as presented in Figure 16, MARVEL prompts the end-user for input data related to the driving cycle and the battery.

Drag Coefficient	0.22
Frontal Area, m ²	1.8
Rolling Resistance	0.005
Driveline Efficiency, %	90
Motor/Generator Efficiency, %	90
Charger Efficiency, %	87.5
Regenerative Braking, Eff., %	50
Acceleration 0-96 kmph, s	13
Top Speed, kmph	120
Grade, %	0.0
Max. Electric Only Range, km	120
Average Daily Range, km	88
Annual Mileage, km	22890
Payload, kg	136
Vehicle Life, yr	12
Vehicle Use, days/yr	260
Annual Interest Rate, %	6.5
Annual Price Escalation, %	1.0
Electricity Rate, \$/kWh	0.06(LA-92); 0.15 (ECE-15)
Gasoline Price, \$/Gallon	1.5(LA-92); 6.0(ECE-15)
Battery State-of-Art Specific Energy @ C/3, Wh/kg	45
Battery Specific Power, W/kg	165 @50%DOD* 150@80%DOD*
Battery State-of-Art Specific Power, W/kg	158@80%DOD
Battery Fixed Cost, C _{BT} , \$	500
Battery Energy Cost, C _{BE} , \$/kWh	96
Battery Power Cost, C _{BP} , \$/kW	18
Battery Life @ 100%DOD, cycles	280
Battery Efficiency, %	78

* At C/3 specific energy of 45 Wh/kg. Many other values for the components are specified as inputs to the models but are not listed here.

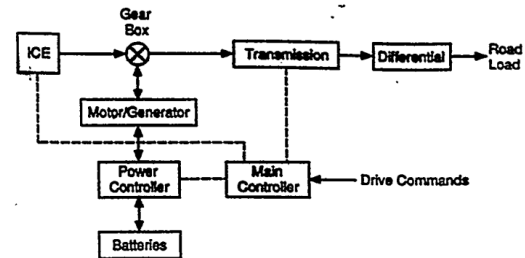


Fig. 16. MARVEL - input data and configuration

As illustrated in Figure 17, the output is shown in a table. In the following example, the table can be used to compare the results in terms of the driving cycle and optimization criteria (life-cycle cost vs. fuel efficiency).

Due to the core being built on FORTRAN, for the same reasons mentioned regarding the JANUS tool, it results in a technologically antiquated tool since its language is outdated and limiting programming-wise.

2.2 Data presentation

Besides the state-of-the-art revolving around the simulation tools, since the proposed solution involves a web client serving as a dashboard for the simulations, it is also essential to consider and analyze similar examples already available, such as fleet management systems.

Driving Cycle	LA-92		ECE-15	
	Life-Cycle Cost	Fuel Efficiency	Life-Cycle Cost	Fuel Efficiency
ICE Size, kW	44.7	41.0	14.9	14.9
Battery Wt., kg	254	254	250	144
Battery Specific Energy @C/3 Rate, Wh/kg	150	155	150	150
Battery Specific Power @80%DOD, W/kg	45.0	41.5	45.0	45.0
Battery kWh@C/3 Rate	11.4	10.5	11.3	6.49
Battery kW@80%DOD	38.1	39.5	37.5	21.6
Battery Max. DOD, %	60	60	65	100
Battery Ave. DOD, %	44	44	47.7	73.4
Battery Life, cycle (yr)	1015(3.9)	1015(3.9)	932(3.6)	517(2.0)
Motor Capacity, kW	33.9	37.6	48.0	43.8
Gross Vehicle Wt., kg	1587	1587	1565	1442
Ave. Energy requirement @ wheels ^{**} , Wh/km-kg	0.059	0.059	0.04	0.04
Peak Power Demand @ wheels, W/kg	32.0	32.0	32.3	32.3
Vehicle Cost, \$	15431	15472	15080	13889
Initial Battery Cost, \$	2282	2222	2257	1512
Replacement Battery Cost, \$	4040	3934	4501	6577
Annual Electricity Consumption, kWh	2612	2408	2695	2472
Annual Gasoline Consumption, gal	169	180	54	60
Annual Energy Costs, \$	410	414	728	731
Fuel Efficiency @ Primary Source [*] , Btu/mile	3723	3664	2675	2556
Life-Cycle Cost, \$/km	0.143	0.144	0.159	0.163

* Assumed primary energy sources (overall efficiency for energy production processes, %) [10]:
 Petroleum to gasoline (83.1)
 Coal to electricity (29.9)

** Including energy recovery from regenerative braking

Fig. 17. MARVEL - simulation results

That is, the web client addressed in this solution will consist of a dashboard with the capabilities to browse/analyze simulation data and its metrics, alongside the execution of a set of actions. Since this context involves cars, their travels, and their charging periods, it will resemble dashboards designed for contexts such as fleet management systems. In fact, there are some examples found on the market, namely Chevin² (shown in Figure 18), Fleetio³ (represented in Figure 19) and GFI Systems⁴ (showcased in Figure 20). Their UIs are designed following a Master-Detail user interface (UI) design approach with plenty of data and metrics related to the selected object.

Furthermore, as a whole, depending on its nature, data can be represented in many forms. When it comes to data (actual and simulation data) such as weather data, traffic data, pollution data, health data, and country statistics, one of its possible representations is through a map to make it more intuitive and easier on the eyes. Several examples can be found, such as the Johns Hopkins COVID-19 Dashboard⁵, illustrated in Figure 21.

²<https://www.chevinfleet.com>

³<https://www.fleetio.com>

⁴<https://www.gfisystems.ca>

⁵<https://coronavirus.jhu.edu/map.html>



Fig. 18. Chevin



Fig. 19. Fleetio



Fig. 20. GFI Systems

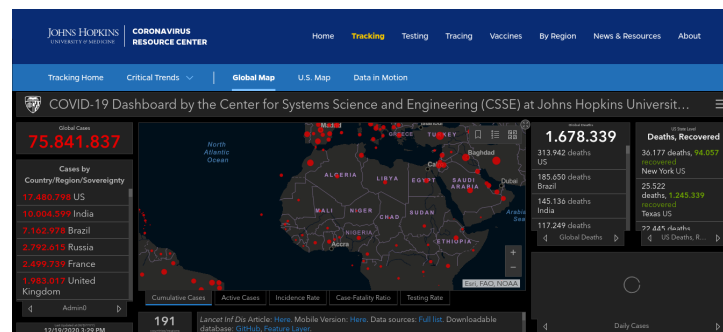


Fig. 21. Johns Hopkins COVID-19 Dashboard

Another example can be found in the SMILE Tukxis administrative dashboard, where the administrative users can consult the travel data and the representation of each travel route in a OpenStreetMaps map (based on the GPS coordinates stored during the travel itself), as seen in Figure 22.

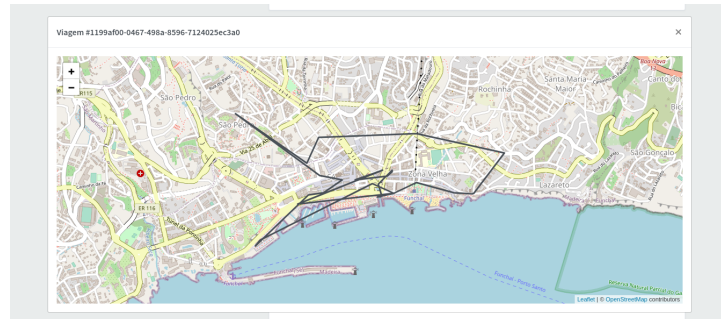


Fig. 22. SMILE Tukxi Dashboard

Regarding our case study, as similar to the examples mentioned above, the travel data is also suitable to its representation on a map. In regards to the Tukxi data, we have the GPS coordinates registered during the travels. Having those coordinates, similarly to the process presented in [8] (whose examples can be observed in Figure 23), we can convert them to a routable road network, generate its corresponding graph, and render it.

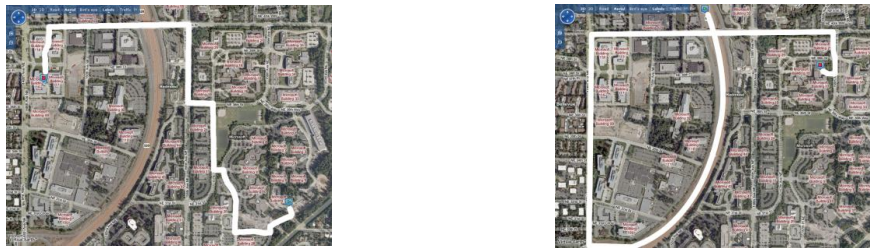


Fig. 23. Travel route example

2.3 Simulators' ideal architecture

Architecturally speaking, we have concluded from the analysis presented above that the reviewed work have adopted a more closed and rigid architecture. Consequently, it makes them hard to adapt to other contexts, affecting their extensibility and scalability.

Therefore, a monolithic approach is not ideal for this kind of simulator. Instead, a microservice[36] approach is more suitable since it results in higher scalability, maintainability, and versatility (in terms of programming languages used in the several microservices). The comparison of both paradigms is illustrated in Figure 24.

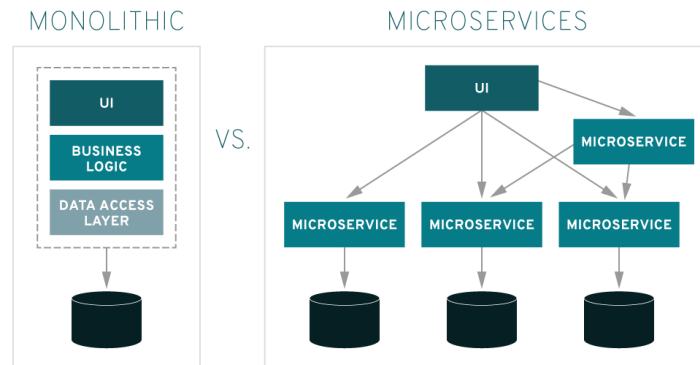


Fig. 24. Monolithic approach vs. Microservice approach

2.4 Conclusions and Solution

Considering the study performed in the state-of-the-art regarding simulation platforms for the energy field, we can summarize each platform as presented in Table 2.

As a result of its analysis, several points were taken into account for the functional and technical aspects of the solution discussed in this thesis.

Most of the tools reviewed before were strict and closed (by adopting closed design patterns). Therefore, it was decided that this solution would not only adopt **open design patterns and approaches** but also be developed as a fully **open-source** program.

Additionally, most tools out there involved MATLAB (partially or fully), which lead to a set of proprietary and inflexible solutions. On the other hand, SimSES was built in Python, leading to more state-of-the-art and modern implementation. Therefore, our approach will be written in **Python**, granting a more open development built upon a more up-to-date programming language with more capabilities (technically speaking).

As observed in SAM, it allowed the creation of custom enhancements (in a set of possible programming languages), which favored the system from the extensibility standpoint. Apart from this, the structural approach presented in JANUS consisted of separating each vehicle component in a separate subroutine. On that account, a similar approach will be used for this solution, making the codebase more readable and more maintainable. Furthermore, V-Elph allowed the usage of user-defined/customized models. Consequently, based on the points mentioned before, apart from adopting Python as the programming language, in terms of architecture, this solution will adopt a **microservice approach**. This will acquire the solution bigger scalability, supporting smaller/easier deployments and, most importantly, allowing a bigger versatility and technical agnosticism in terms of programming languages (being that each component can be built upon any, independently).

Besides that, the vast majority of the tools above included some form of not-only textual outputs but also in a graphical format. Hence, the **graphical representation of the simulations** will be added as another inclusion to this thesis.

To finalize, we observed that most of the reviewed work consists of desktop applications. This type of application often requires more time on setup/maintenance and specific software/hardware components

Tool	Application	Technology	Publication year	Summary
PerMod	BESSs	MATLAB	2020	Simulation and analysis of battery energy storage systems' performance energy-wise
SimSES	BESSs	Python (MATLAB initially)	2019	Open-source techno-economic simulator of stationary energy storage systems
SAM	Renewable energy	C/C++	2018	Versatile open-source renewable energy projects' simulator that also includes the examination of their techno-economical factors
FreeGreenius	Renewable energy	n/a	2018	Economic simulation of renewable power plants, taking into account meteorological data, and several economic costs and variables
PSIM	Automotive systems	n/a	2004	Simulation of power electronic converter and motor drives (including automotive systems)
JANUS	Automotive systems	FORTRAN	1985	Simulation of vehicles regarding design, performance, and efficiency
V-Elph	Automotive systems (EVs + HEVs)	MATLAB	1999	Simulation tool for analysis and comparison of EV and HEV setups and/or energy management strategies
BLAST	BESSs	n/a	2014	Prediction of EV batteries' response, based on its properties and history
SIMPLEV	Automotive systems (EVs)	BASIC	1991	Parametric simulations and studies on EVs
ADVISOR	Automotive systems	MATLAB	1994	Parameterized vehicle simulation
MARVEL	Automotive systems (HEVs)	FORTRAN (PL/I initially)	1995	Analysis of HEV systems, according to a specific driving cycle and optimization criteria

Table 2. Related work - Summary

to be runnable. However, by **adopting a web approach** for this solution, it assures the researchers that they can easily access it through any device with minimal to no setup while also reducing development costs and guaranteeing consistency across all kinds of systems.

3 SOLUTION

Motivated by the analysis of the literature above (2.4), in this section, the proposed solution will be addressed. Firstly, we will present the process of requirements gathering. Afterwards, the architecture of the proposed solution is explained, including its description, its modeling, the development tools used, and its representation in diagrams (such as class diagrams and event diagrams).

3.1 Elicited requirements

Before the development itself, the requirements (both functional and non-functional) were gathered alongside the investigation team associated with this project. They are listed in Table 3.

Requirement	Requirement description
R1	The system must integrate third-party consumers as data models
R2	The system must be capable of being continuously modified/extended, e.g., with new data models
R3	The user shall be able to configure the execution of simulations
R4	The user shall be able to interact with the simulations
R5	The system must expose the simulation data in real-time
R6	The system must store its data in a database
R7	The user shall be able to export the database
R8	The system must expose external REST API endpoints for data consumption and interactions with the simulator
R9	The system must be capable of sending notifications concerning the simulations to a third-party
R10	The user shall be able to browse through data of previous simulations

Table 3. Elicited requirements

3.2 Modelling the solution

The main goal of the work presented in this sub-section consists of creating the data models to use in the case study of this thesis. Even though our contribution is not focused on a particular case study, the process presented below illustrates how to integrate data from different data sources into the proposed platform. This demonstration will serve as another proof regarding the flexibility of the solution.

As previously mentioned, the case study modelled on this solution is based on the data gathered from the SMILE project. In the remainder of this sub-section (and its sub-sub-sections), the modeling process of different aspects of the EV pilot will be described.

In practical terms, the process involved here consisted of the latter:

- (1) **Extracting** the data from its data source
- (2) **Processing** the data in an analysis tool
- (3) Forming the **mathematical models'** formulas

Technically speaking, the gathered data came from the SMILE Tukxis' API located at 'https://smile.prisma.com/tukxi/api/', which contains information related to the available Tukxis (e.g., its drivers, its travels). Its endpoints are listed in Table 4.

Endpoint	Method	Description
/auth/token	GET/POST	Get the access token
/drivers	GET	Get the list of drivers
/driver/{driver_id}/actions	GET	Get the actions history of a driver (e.g., start/end of a charging period, pick-up, drop-off)
/driver/{driver_id}/travels	GET	Get the travels of a driver
/plugs/	GET	Get the list of plugs
/plug/{plug_id}/actions	GET	Get the actions history of a plug
/plug/{plug_id}/state	GET	Get the state of a plug
/plug/{plug_id}/state/{state}	POST	Set a particular state for a plug
/plug/{plug_id}/historical-consumption/{start}/{end}/{non_0}	GET	Get the energy consumption history of a plug
/cars	GET	Get the list of cars
/cars/status	GET	Get the status of each car
/car/{car_id}/action/{action_type}	POST	Set a particular action for a plug
/car/{car_id}/actions	GET	Get the actions history of a car
/car/{car_id}/travel/start	POST	Start a travel for a car
/car/{car_id}/travel/{travel_id}/points	POST	Register a GPS coordinate for a certain travel
/car/{car_id}/travel/{travel_id}/end	POST	End a travel for a car
/car/{car_id}/travels	GET	Get the list of travels of a car
/routes	GET	Get the list of Tukxi routes
/travel/{travel_id}/points	GET	Get the registered points of a travel

Table 4. SMILE Tukxi API endpoints

The data came out to be categorized into the following data models:

- **Travels and battery consumption**
- **Charging**
- **Affluence**

The first data model embodies the information related to the traveled distances during the Tukxis' routes, alongside their battery consumption. In the same way, the second one involves the data regarding the Tukxis' charging periods - their duration, their peak value (in terms of its toll on the electrical network). The third model comprises the information linked to the travels' affluence during the day.

After the development of the models mentioned above, our solution allows the researchers to simulate the whole smart-charging process. In other words, it can simulate the three subprocesses behind it: the

travel process, the charging process, and the determination of the travel rate (depending on the time of day).

The following subsections present the procedure previously described for each data model.

3.2.1 Travels and battery consumption

As mentioned above, this data model is composed of the traveled distances and their battery consumption. For its modelling, the procedure was the following:

- (1) Fetching the list of **cars** - via API endpoint '/cars'
- (2) Fetching the **travels of each car** - via API endpoint '/car/{car_id}/travels'
- (3) **Filtering** the travels, by the following criteria:
 - Initial battery > 0
 - Final battery > 0
 - Initial battery >= Final battery
 - Battery consumption > 0
 - Traveled distance was registered (client-side distance > 0 OR server-side estimated distance > 0)
 - Traveled distance > 1 km
 - Traveled distance < 40 km
- (4) Calculating the **average traveled distance** in km and its standard deviation
- (5) Calculating the **average battery consumption** per km and its standard deviation

The gathering and processing of this data lead up to the following calculations shown below in Table 5.

Calculation	Value	Standard deviation
Average travel distance (km)	~12.421 km	~8.967 km
Average battery consumption (per km)	~0.504	~0.676

Table 5. Travels - results

Concerning the travel distance, its calculations paved the way to the formula below:

$$t_{dist} = avg_{dist} \pm std_{dist}$$

$$t_{dist} = 12.421 \pm 8.967 \quad (1)$$

t_{dist} Traveled distance (km)

avg_{dist} Average traveled distance (km)

std_{dist} Standard deviation of the average traveled distance (km)

The formula above is illustrated in Figure 25.

$$f_{bat} = i_{bat} - t_{dist} \times (avg_{cons} \pm std_{cons})$$

$$f_{bat} = i_{bat} - t_{dist} \times (0.504 \pm 0.676) \quad (2)$$

f_{bat} Final battery

i_{bat} Initial battery

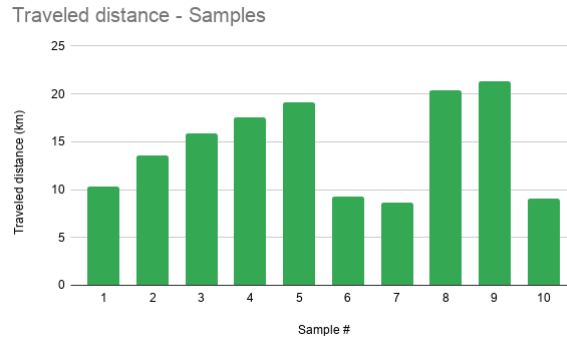


Fig. 25. Travel distance sample

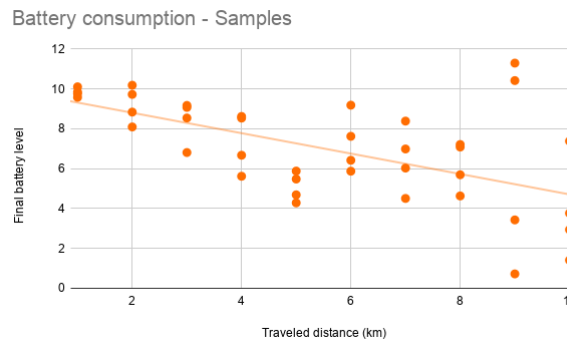


Fig. 26. Travel battery consumption sample

t_{dist} Traveled distance (km)

avg_{cons} Average battery consumption (per km)

std_{cons} Standard deviation of the average battery consumption (per km)

Figure 26 presents a couple of travel samples applied to the formula above (with a supposed 10 (that is, 100%) of initial battery level).

3.2.2 Charging

Regarding this data model, it comprises the data of the charging periods (their duration and their peak value). The modelling process went by the next set of steps:

- (1) Fetching the energy **consumption data** - via API endpoint `/plug/{plug_id}/historical-consumption/{start}/{end}/`
- (2) **Filtering** the energy consumption data, according to the following criteria:
 - Energy consumption > 50 W
 - Battery charged > 0
- (3) Calculating the **average charging period duration** (in minutes) and its standard deviation
- (4) Calculating the **average charging peak value** (in W) and its standard deviation

The calculations' results can be observed in Table 6.

Calculation	Value	Standard deviation
Average charging period duration (minutes)	~142.889	~44.898
Average charging period peak value (W)	~2666.817	~221.847

Table 6. Charging - results

Subsequently, the calculations resulted in the formulas below:

$$d_{cperiod} = avg_{d_{cperiod}} \pm std_{d_{cperiod}}$$

$$d_{cperiod} = 142.889 \pm 44.898 \quad (3)$$

$d_{cperiod}$ Charging period duration (minutes)

$avg_{d_{cperiod}}$ Average charging period duration (minutes)

$std_{d_{cperiod}}$ Standard deviation of the average charging period duration (minutes)

$$pk_{cperiod} = avg_{pk_{cperiod}} \pm std_{pk_{cperiod}}$$

$$pk_{cperiod} = 2666.817 \pm 221.847 \quad (4)$$

$pk_{cperiod}$ Peak value of a charging period (W)

$avg_{pk_{cperiod}}$ Average peak value of a charging period (W)

$std_{pk_{cperiod}}$ Standard deviation of the average peak value of a charging period (W)

These formulas are exemplified in Figures 27 and 28.

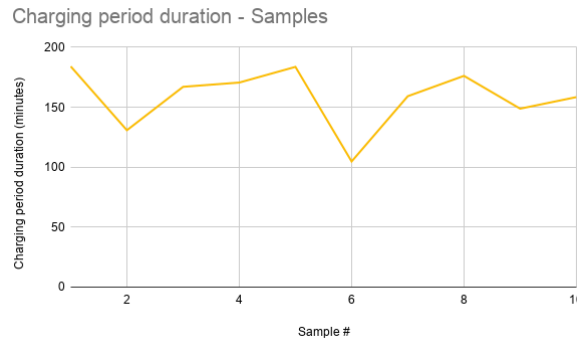


Fig. 27. Charging period duration sample

3.2.3 Affluence

This last model embodies the travel affluence data of a given hour of the day. It reused the data extracted regarding the travels, in which we gathered the travel affluence per hour of day. By looking at that same data, we can state that the travels occur mainly in the early morning and the early afternoon. This affluence is shown with more detail in Table 7 and illustrated in Figure 29.

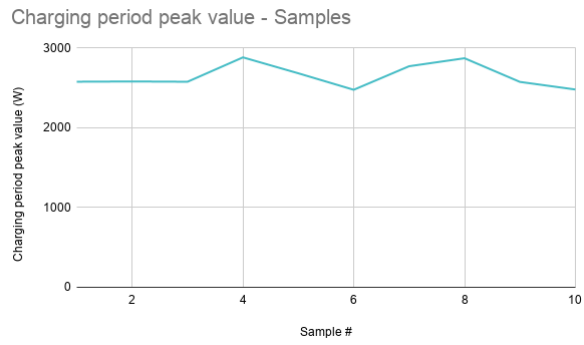


Fig. 28. Charging peak value sample

Time of day	Travel count
0h00-0h59	0
1h00-1h59	0
2h00-2h59	0
3h00-3h59	0
4h00-4h59	0
5h00-5h59	0
6h00-6h59	0
7h00-7h59	0
8h00-8h59	10
9h00-9h59	20
10h00-10h59	17
11h00-11h59	17
12h00-12h59	8
13h00-13h59	9
14h00-14h59	7
15h00-15h59	11
16h00-16h59	10
17h00-17h59	7
18h00-18h59	3
19h00-19h59	3
20h00-20h59	3
21h00-21h59	0
22h00-22h59	1
23h00-23h59	1

Table 7. Affluence - results

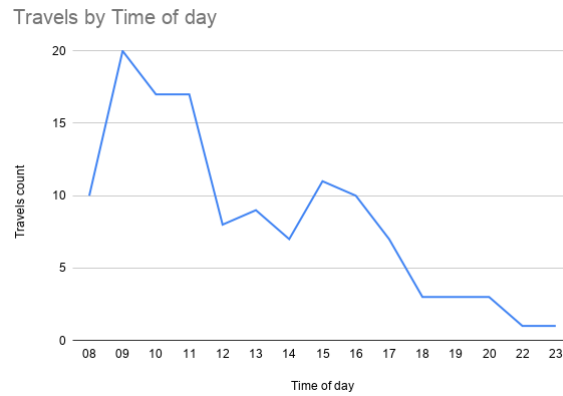


Fig. 29. Affluence - results' representation

3.2.4 Summary and observations

To recapitulate, the data models can be summarized in Table 8.

Variable	Formula
Average travel distance (km)	$t_{dist} = 12.421 \pm 8.967$
Average battery consumption (per km)	$f_{bat} = i_{bat} - t_{dist} \times (0.504 \pm 0.676)$
Average charging period duration (minutes)	$d_{cperiod} = 142.889 \pm 44.898$
Average charging period peak value (W)	$pk_{cperiod} = 2666.817 \pm 221.847$
Affluence	0h00-7h59 = 0 8h00-8h59 = 10 9h00-9h59 = 20 10h00-11h59 = 17 12h00-12h59 = 8 13h00-13h59 = 9 14h00-14h59 = 7 15h00-15h59 = 11 16h00-16h59 = 10 17h00-17h59 = 7 18h00-20h59 = 3 21h00-21h59 = 0 22h00-23h59 = 1

Table 8. Data models - Summary

Also, for matters of simplification, the following assumptions were made:

- The travels will be roundtrip (they start and end in the same location)

- If a car ends up under 2 (that is, 20% battery), it means that then the car will be charged up (based on the charging data model)

Regarding the charging periods, its model was formed using the plug energy consumption history's endpoint since it was difficult to isolate the charging periods (using the cars' actions endpoint) and also because the history endpoint leads to less variability.

Besides that, the travels extracted from the API will be used as templates for possible travels to be done in the simulator, and their trajectories will be represented accordingly in the simulator's map.

Then again, the data models presented here serve as a simple illustration of the process involving integrating and modeling data from any external data source into the proposed platform in this thesis. That is, reinforcing the concept of flexibility and versatility designed for this solution since it will easily incorporate data from any data source. In other words, if any other area needs to be simulated or if any other data source needs to be considered in the simulations, it can be accomplished by following the process described above.

3.3 Development tools

In terms of the development itself, as mentioned before, its core and its main modules will be built in **Python**⁶, being that its containerization will be made using **Docker**⁷ and that the whole scripting side will be made recurring to **Makefiles**⁸.

Furthermore, **SQLObject**⁹ will be used to handle the connection between the Python objects and its database through Object-relational mapping (ORM) classes.

For the implementation of the models themselves, the following frameworks will be used: **TensorFlow**¹⁰ for their development and training, **Nameko**¹¹ for the construction of the microservices and **RabbitMQ**¹² as a message broker for the microservices.

Moreover, the **websockets**¹³ library will be utilized to serve a WebSocket (WS) between the simulator and its web client.

In addition, we will recur to **Flask**¹⁴ for the formation of the external Representational state transfer (REST) API and to serve the web client's static files.

Regarding the web client, **OpenUI5**¹⁵ will be used as the front-end UI framework.

3.3.1 Core programming language

Again, the solution proposed in this thesis will be constructed using Python. This high-level and object-oriented programming language provides plenty of libraries and capabilities such as data structures and classes. Moreover, this language allows a modernized, intuitive, and productive implementation for this

⁶<https://www.python.org>

⁷<https://www.docker.com>

⁸<https://www.gnu.org/software/make>

⁹<http://www.sqlobject.org>

¹⁰<https://www.tensorflow.org>

¹¹<https://nameko.readthedocs.io>

¹²<https://www.rabbitmq.com>

¹³<https://websockets.readthedocs.io>

¹⁴<https://flask.palletsprojects.com>

¹⁵<https://openui5.org/>

thesis[38]. In terms of this thesis, Python will compose the solution's foundation - its main modules and its integration with the remainder of the used frameworks to incorporate the communication protocols between the solution and the microservices involved.

```

384     endless_query = self,
385     endless_query.st: f clone()
386     self._all_patches: f compute_percentile(percentage)
387     return self._all
388         v end_position
389     def compute_percenti: f generate_patches()
390     """
391     Returns a Positi: f get_all_patches(dont_use_cache=False)
392     through the larg: f get_end_position()
393         f get_start_position()
394     @param percentage: v inc_extensionless
395     """
396     all_patches = se: v path_filter
397     return all_patch: v root_directory
398         int(len(all_): compute_percentile(self, percentage)
399         ], start_position
400     Returns a Positi: Returns a Position object that represents percentageN-far-of-the-way
401     through the larg: through the larger task, as specified by this query.
402     def generate_patches: """
403     Generates a list @param percentage a number between 0 and 100.
404     self.root_directory
405     that satisfy the given conditions given

```

Fig. 30. Python - snippet

Alongside Python, the **pipreqs**[17] Python module is used in order to generate the Python dependencies file named `requirements.txt` based on the imports defined in the source code, easing the processing revolving around the installation of the dependencies in Python projects.

3.3.2 Machine learning

In the same way, TensorFlow is an open-source machine learning platform known for being able to develop and train models from a high-level standpoint, easing up the creation of models and having a flexible architecture[45]. Moreover, in the context of this thesis, TensorFlow will be used to create and train the different data models that will be consumed as microservices in this solution while incorporating machine learning and neural networks.

```

import tensorflow as tf
import tensorflow_transform as tft
import tensorflow_transform.beam as tft_beam

def preprocessing_fn(inputs):
    x = inputs['x']
    y = inputs['y']
    s = inputs['s']
    x_centered = x - tft.mean(x)
    y_normalized = tft.scale_to_0_1(y)
    s_integerized = tft.compute_and_apply_vocabulary(s)
    x_centered_times_y_normalized = x_centered * y_normalized
    return {
        'x_centered': x_centered,
        'y_normalized': y_normalized,
        'x_centered_times_y_normalized': x_centered_times_y_normalized,
        's_integerized': s_integerized
    }

```

Fig. 31. TensorFlow - snippet

3.3.3 *Microservices development*

Nameko is a framework used for the creation of microservices without having to worry with its low-level logic while having the built-in support for several communication features (e.g., HTTP GET/POST, messages, event dispatching, event listening)[29]. On this thesis, it will be used to construct the individual microservices for each data model.

```
# helloworld.py
from nameko.rpc import rpc

class GreetingService:
    name = "greeting_service"

    @rpc
    def hello(self, name):
        return "Hello, {}".format(name)
```

Fig. 32. Nameko - snippet

3.3.4 *API development / Static file serving*

Flask consists of a Web Service Gateway Interface (WSGI) designed to ease the production of APIs, on a largely open and extensible environment[13], while also providing the possibility of serving static files. Regarding the simulator, this framework will be used as a means to create and maintain the solution's external API that will contain several endpoints with simulation functionalities. Considering the web client, Flask will be used to serve the static files required for the web client itself.

```
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route('/')
5 def hello_world():
6     return 'Hey, we have Flask in a Docker container!'
7
8
9 if __name__ == '__main__':
10    app.run(debug=True, host='0.0.0.0')
```

Fig. 33. Flask - snippet

3.3.5 *Real-time communication*

In order to implement the communication between the web client and the simulator itself, the websockets Python library will be used. It provides Python with a high-level and simple API to build WebSocket servers and/or WebSocket clients[46]. Concerning this thesis, this library will serve a WebSocket to be consumed by the web client.

```
#!/usr/bin/env python

import asyncio
import websockets

async def echo(websocket, path):
    async for message in websocket:
        await websocket.send(message)

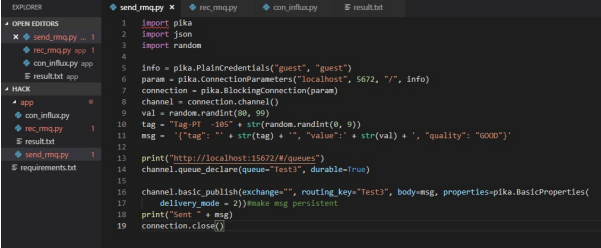
start_server = websockets.serve(echo, "localhost", 8765)

asyncio.get_event_loop().run_until_complete(start_server)
asyncio.get_event_loop().run_forever()
```

Fig. 34. websockets - snippet

3.3.6 Message brokers

In terms of message brokers, RabbitMQ[39] is one of the most popular in the industry, delivering a lightweight solution that supports many communication protocols and provides high scalability and high availability. It serves as a message-oriented middleware based on a Advanced Message Queuing Protocol (AMQP), an open standard that provides interoperable communication between systems independently of the platforms in question[3]. Through Remote Procedure Call (RPC) communication, the RabbitMQ framework forms a message broker between the several microservices available in this system.



```
1 import pika
2 import json
3 import random
4
5 info = pika.RabbitCredentials("guest", "guest")
6 param = pika.ConnectionParameters("localhost", 5672, "/", info)
7 connection = pika.BlockingConnection(param)
8 channel = connection.channel()
9 val = random.randint(0, 99)
10 tag = "tag-P1-" + str(random.randint(0, 9))
11 msg = '{"tag": "' + str(tag) + '", "value": ' + str(val) + ', "quality": "GOOD"}'
12
13 print("http://localhost:15672/#/queues")
14 channel.queue_declare(queue="Test3", durable=True)
15
16 channel.basic_publish(exchange="", routing_key="Test3", body=msg, properties=pika.BasicProperties(
17     delivery_mode = 2))#make msg persistent
18 print("Sent - " + msg)
19 connection.close()
```

Fig. 35. RabbitMQ - snippet

3.3.7 Containerization

One of the most prominent containerization platforms is Docker[12] since it enables the creation of isolated and straightforward software units for applications without any dependency on the environment itself while maintaining consistency for the whole lifecycle revolving around its development and deployment. These software units are named Docker containers. In the scope of this thesis, Docker will be used to set up the workflow for the whole solution, setting up a container for its core, a container for the Web client, a container for the gateway (used to centralize the communication to the microservices) and a container for each data model.

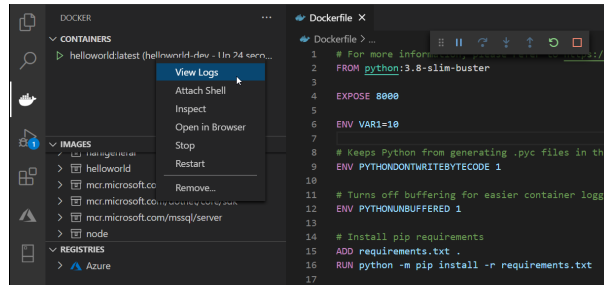


Fig. 36. Docker - snippet

Furthermore, in order to streamline the effort involved in the configuration steps for each Docker container, this thesis will recur to the **Pack**[35] tool. Maintained by the Cloud Native Buildpacks project, this tool uses buildpacks to generate the runnable images (and all their configuration) solely based on the source code found. As a result, the process of preparing/configuring, and building the Docker images will be fully automated by such buildpacks.

3.3.8 Front-end development

Considering the simulator's Web client, it will be based upon the OpenUI5[34] framework, built on top of technologies such as Javascript, HTML5, CSS, and XML. This framework brings up many benefits, such as its responsiveness, its feature-rich UI controls, its consistent user experience (UX), the adoption of enterprise-level development concepts/principles, and following a Model-View-Controller architecture (MVC).

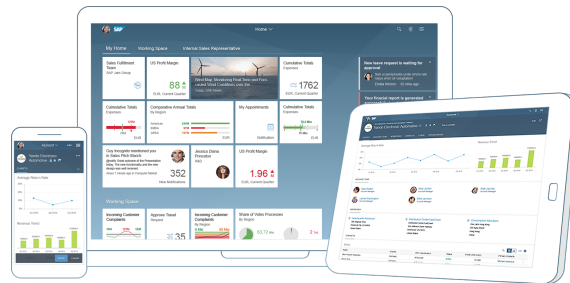


Fig. 37. OpenUI5

In addition, for the renderization of the simulation stats, this web client will recur to the Chart.js[9] framework. It includes a responsive and interactive set of possible charts to be rendered on any web page.

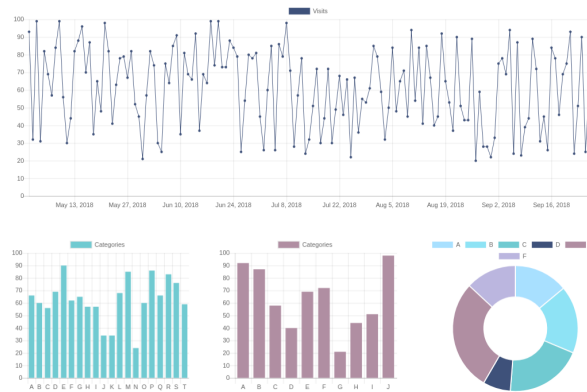


Fig. 38. Chart.js

3.3.9 Scripting

Concerning the steps involved in processes such as the build for each Docker container, we will recur to Makefiles[20], that eases up the preparation of such steps with lesser code and higher flexibility when compared to the traditional bash scripting.

```

1  deps_install := $(CURDIR)/build/last-install-time
2  pkg_lib_foo  := $(CURDIR)/build/foo-1.0.0.tgz
3  pkg_lib_bar  := $(CURDIR)/build/bar-1.0.0.tgz
4  pkg_lib_baz  := $(CURDIR)/build/baz-1.0.0.tgz
5  pkg_alpha   := $(CURDIR)/build/alpha-1.0.0.tgz
6  pkg_omega   := $(CURDIR)/build/omega-1.0.0.tgz
7
8  .PHONY: all handlers libs
9
10 all: libs handlers
11
12 handlers: $(deps_install)
13   $(MAKE) -C src/handlers/alpha
14   $(MAKE) -C src/handlers/omega
15
16 libs:
17   $(MAKE) -C src/lib/foo
18   $(MAKE) -C src/lib/bar
19   $(MAKE) -C src/lib/baz
20
21 $(deps_install): $(pkg_lib_foo) $(pkg_lib_bar) $(pkg_lib_baz)
22   @if [ "$(pkg_lib_foo)" = "$(findstring $(pkg_lib_foo),$*)" ]; then \
23     cd $(CURDIR)/src/handlers/alpha && npm i $(pkg_lib_foo); \
24   fi
25   @if [ "$(pkg_lib_bar)" = "$(findstring $(pkg_lib_bar),$*)" ]; then \
26     cd $(CURDIR)/src/handlers/alpha && npm i $(pkg_lib_bar); \
27   cd $(CURDIR)/src/handlers/omega && npm i $(pkg_lib_bar); \
28   fi
29   @if [ "$(pkg_lib_baz)" = "$(findstring $(pkg_lib_baz),$*)" ]; then \
30     cd $(CURDIR)/src/handlers/omega && npm i $(pkg_lib_baz); \
31   fi
32   @touch $(deps_install)
33

```

Fig. 39. Makefile

3.3.10 Object-relational mapping

Regarding the data persistence on a database, SQLAlchemy is a Python library made as a high-level interface that connects the Python classes involved in a project to an actual database table and its CRUD operations. The object attributes of these classes correspond to database columns and that their instances correspond to database rows[44]. Accordingly, it will be used in this thesis to store the simulation data in a database.

```
from sqlalchemy import *
sqlhub.processConnection = connectionForURI('sqlite://:memory:')

class Person(SQLObject):
    fname = StringCol()
    mi = StringCol(length=1, default=None)
    lname = StringCol()

Person.createTable()
```

Fig. 40. SQLAlchemy - snippet

3.4 Architecture

The architecture involved in this solution consists of the following modules (separated by Docker containers, run independently, as illustrated in Figure 65):

- **Simulator**
- **Gateway**
- **Data models**
 - Travel - Affluence
 - Travel - Distance
 - Travel - Duration
 - Travel - Final battery level
 - Charging period - Duration
 - Charging period - Energy spent
- **Web client**

3.4.1 Simulator

The simulator's core is implemented in the **Simulator** class, that serves as a wrapper class to the main functionalities of this system, such as the start/stoppage of simulations, the WebSocket message receival/sending process, and the database export.

Additionally, the core also contains the following set of helper classes: **DataServer** (that takes care of the exposure of the external REST API endpoints), **ConfigurationHelper** (that wraps the operations concerning the configuration file), **Logger** (that unifies the logging process), **WebhookHelper** (that handles the whole notification process through a Slack Webhook), **DBHelper** (that abstracts both the initialization of the database and the export of its data), **DebugHelper** (that provides some utility functions targeted for debugging purposes), **SingletonMetaClass** (a Python metaclass made for the formation of singleton classes) and **StatsHelper** (that encapsulates the logic involved in the preparation of the simulation statistics).

Furthermore, the data objects involved in the simulation process are implemented in the following classes: **Simulation**, **Car**, **Plug**, **Log**, **Stat**, **Travel** and **ChargingPeriod**. Recurring to ORM classes, each of these objects has its respective model class that inherits the BaseModel class (that inherits from the SQLAlchemy class), that will handle its CRUD operations with the database. Thus, these model classes follow the database structure represented in Figure 41. In other words, for each database table, there is a corresponding subclass of BaseModel, being that its table columns correspond to class attributes and that each instance represents a row of its table.

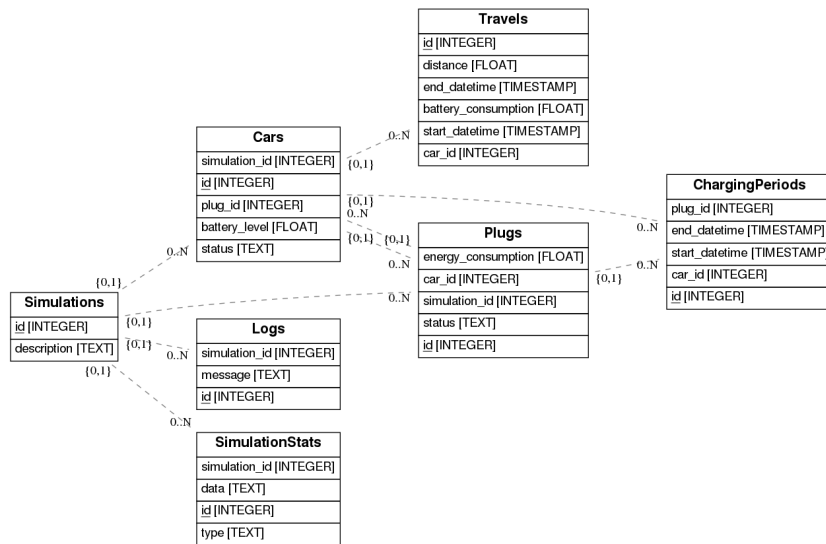


Fig. 41. Simulator - ER Diagram

Concerning their business logic, the model classes have a corresponding subclass of BaseModelProxy, in which this logic will be implemented.

The architecture described above is illustrated in Figure 42.

3.4.2 Gateway

The gateway is used to centralize the communication between the simulator and the data models. In other words, it comes up as a microservice that serves a set of HTTP endpoints that, when called, delegates its logic to its respective data model (via its respective RPC proxy). Afterwards, the corresponding data model will handle the request and return a response accordingly to its business logic.

In technical terms, the gateway service contains the implementation of each endpoint and their associated RpcProxy instances (used to delegate the endpoints' handling).

Its architecture is represented in Figure 43.

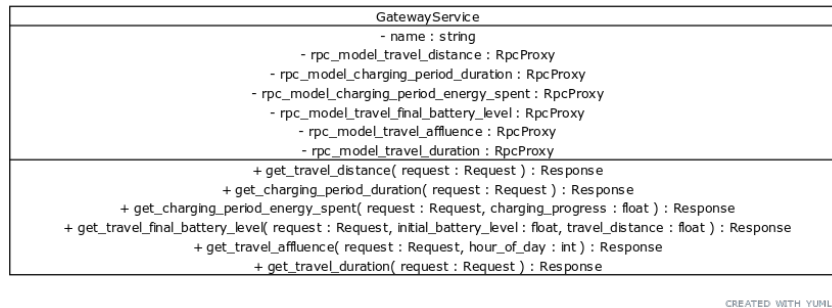


Fig. 43. Gateway component - architecture

3.4.3 Data models

The data models' architecture can be easily described by its name (used to distinguish and instantiate the respective RpcProxy from the gateway) and its clean and encapsulated business logic implementation.

More specifically, regarding the travels' affluence, its model (represented in Figure 44) consists on the calculation of the travel affluence according to a certain hour of day as an input. Equally, its distance

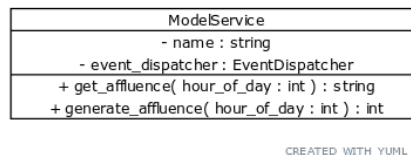


Fig. 44. Travel affluence model - architecture

model (that can be observed in Figure 45) is implemented simply through a random generation of a travel distance. In the same way, the logic involved in its duration model (drawn in Figure 46) plainly revolves around a random generation of a travel duration. Similarly, the model concerning the final battery level of a car in the end of a travel (illustrated in Figure 47) consists on a model that, based on an initial battery level and a travel distance as input, generates a final battery level for the car.

Furthermore, in regards to the charging periods, its duration model's implementation (that can be observed in Figure 48) is described as a plain random generation of the duration value.

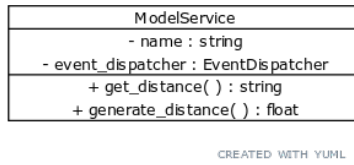


Fig. 45. Travel distance model - architecture

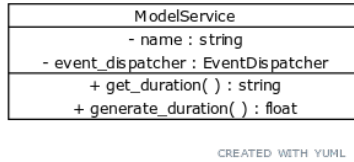


Fig. 46. Travel duration model - architecture

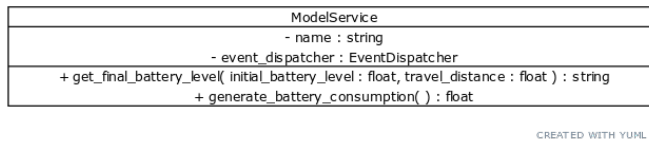


Fig. 47. Travel battery consumption model - architecture

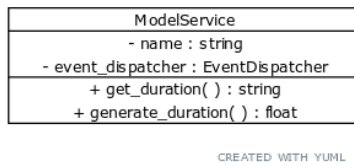


Fig. 48. Charging period duration model - architecture

Concerning its energy expenditure model (drawn in Figure 49) is capable of, based on the charging period's progress (in percentage), randomly generating its energy expenditure value.

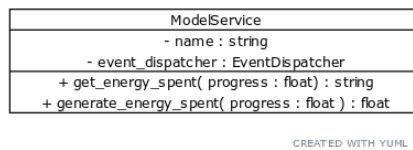


Fig. 49. Charging period energy consumption model - architecture

3.4.4 *Web client*

The web client that will interact with the simulator and consume its data is composed of controllers (inheriting from a base controller with plenty of utility functions), a customized UI control, and two helper classes.

A base controller (named `BaseController`) implemented several common methods regarding functionalities such as UI handling and general data handling. Inheriting this controller, the client has a controller for its core and one for each view present, following a MVC architecture. Each controller handles UI operations such as UI events (such as the click event handling for a given button) and view-specific UI formatting.

Moreover, the UI control `StatsChart` was developed in this thesis to render the charts related to the simulation data gathered.

Likewise, two helper classes (`MessageHelper` and `SocketHelper`) were built to encapsulate both the parsing process of the messages received through the simulator's `WebSocket` and the `WebSocket` handling revolving its connection and messages sent through the client, respectively.

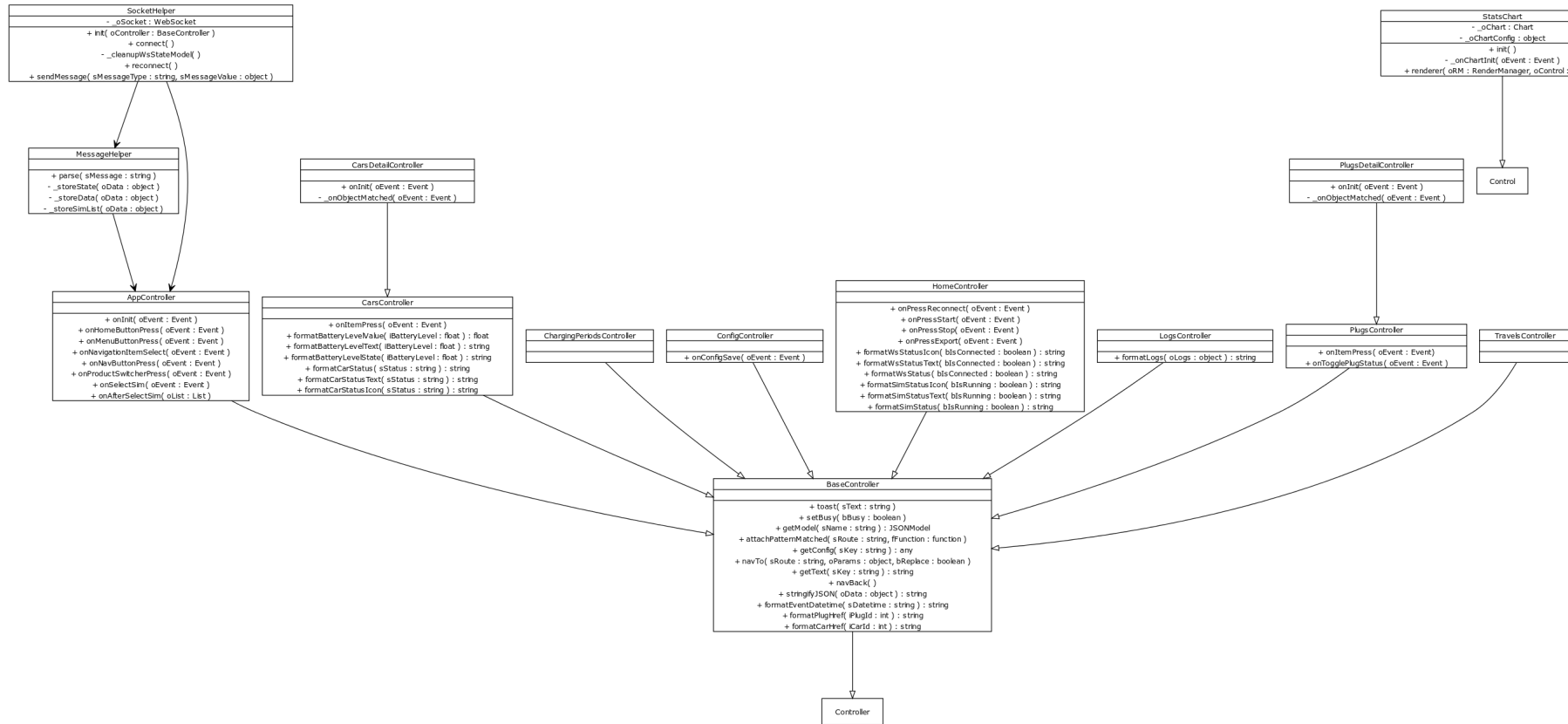


Fig. 50. Client component - architecture

3.5 Functional overview

In this solution we can divide the end-users in two categories: the **web end-users** and the **standard end-users**.

The first set of end-users consists of researchers that use the Web client for their tasks, such as checking up on the simulation status, browsing through simulation data, starting a new simulation, and exporting the whole simulator's data. On the other hand, the second set is composed of users that do not use the Web client and merely execute their actions through the simulator's external REST API.

Figure 51 presents the use cases for each type of user.

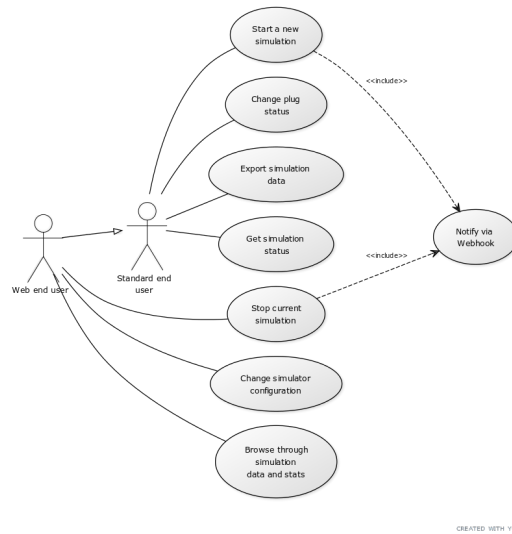


Fig. 51. Use cases

More specifically, the simulator is composed by the following processes: **simulations**, **travels** and **charging periods**. To help understanding them, each is illustrated in flowcharts (Figures 52, 53 and 54, respectively).

In this section, the crucial points regarding the architecture and functionalities of the proposed solution were described. In the following section, the implementation of those same points will be addressed.

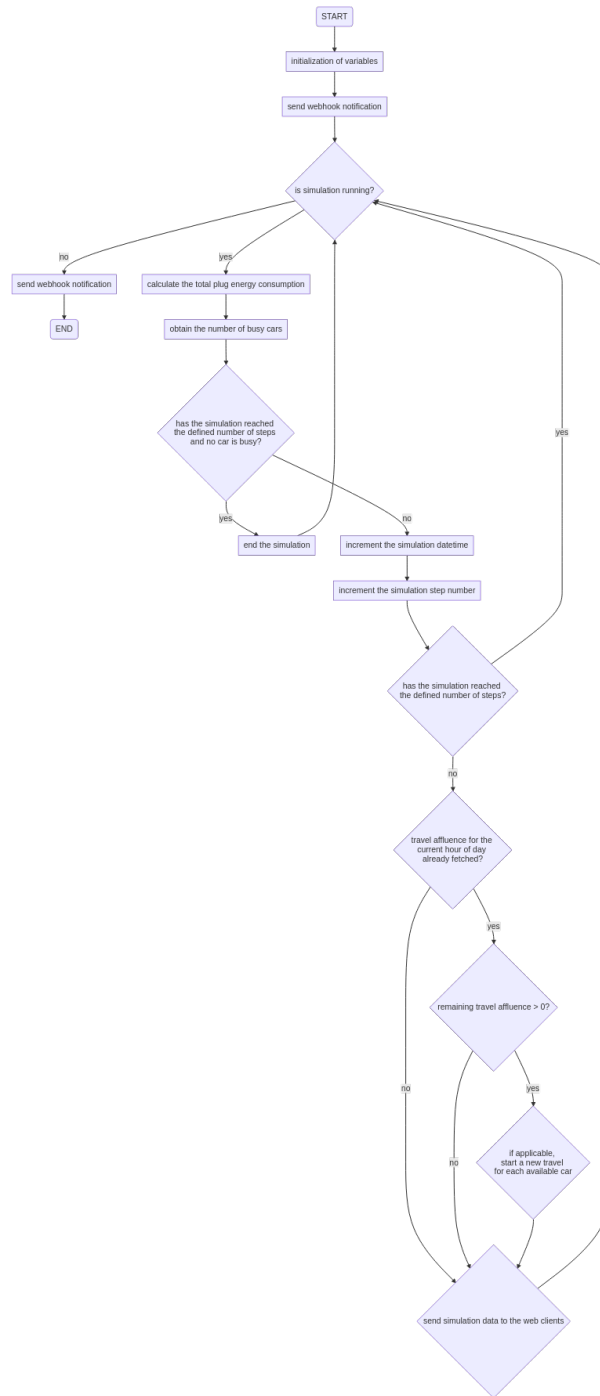


Fig. 52. Simulation flowchart

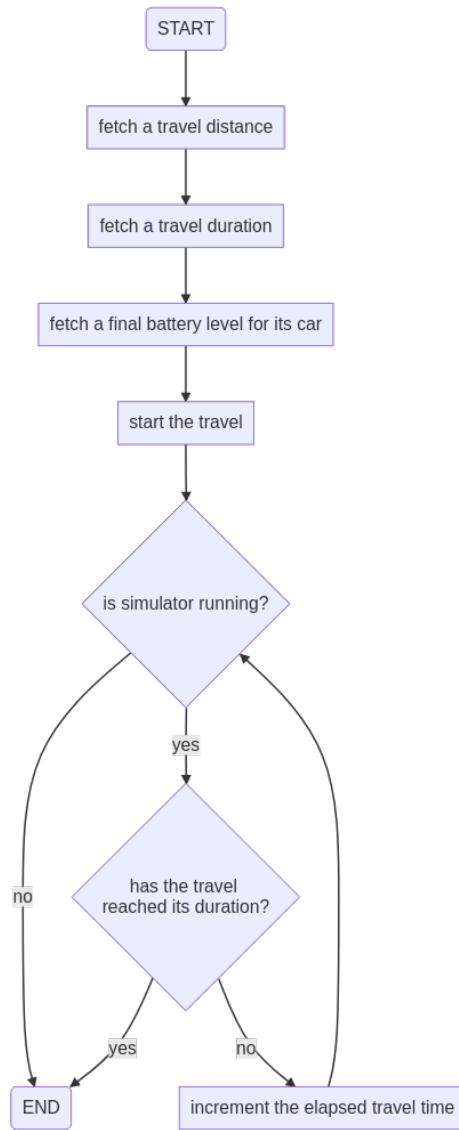


Fig. 53. Travel flowchart

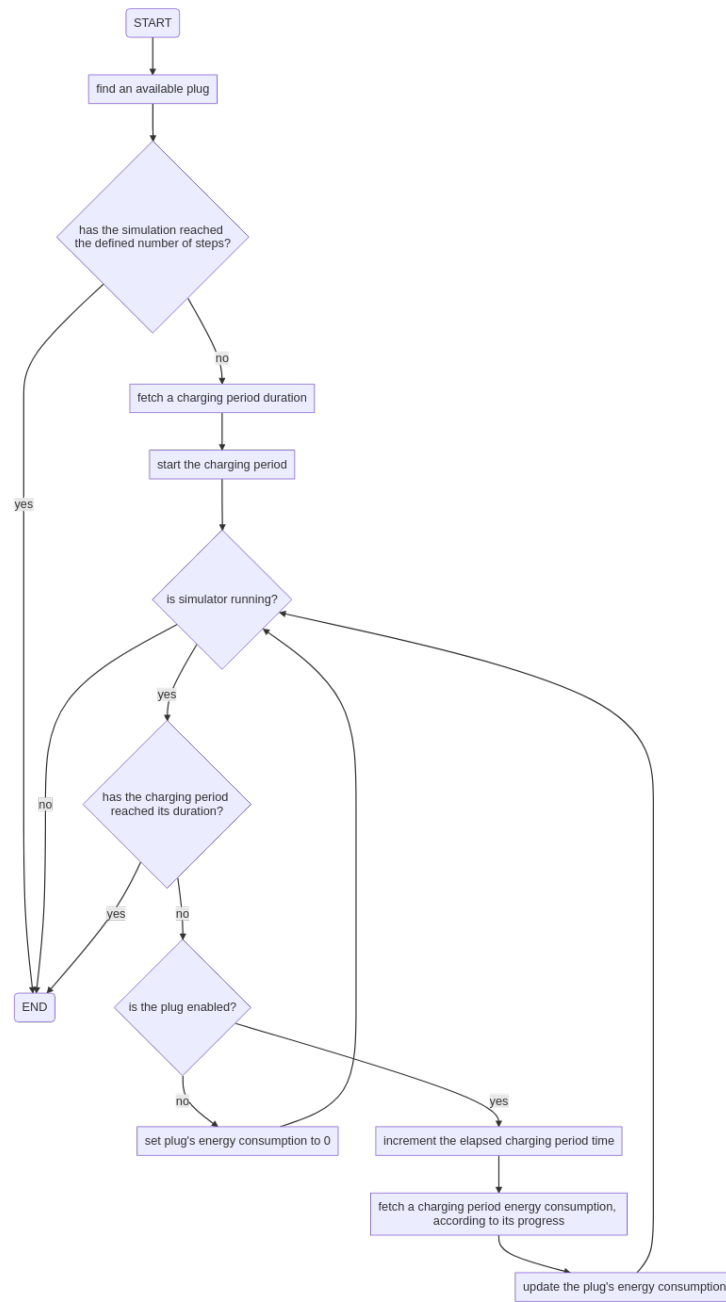


Fig. 54. Charging period flowchart

4 IMPLEMENTATION

In this section, the technical aspects of the developments will be described, including the Docker containers involved, alongside a set of code snippets.

4.1 Git flow

All the developments made for this thesis followed the usual Git flow branching model. In other words, as illustrated in Figure 55, it consisted on three sets of branches: **master** (to store the developments in final/productive state), **develop** (a branch for testing and integration of the developments) and the **dev/XXX** branches (where each new functionality was implemented). In addition, for each container, there is a corresponding Git repository.

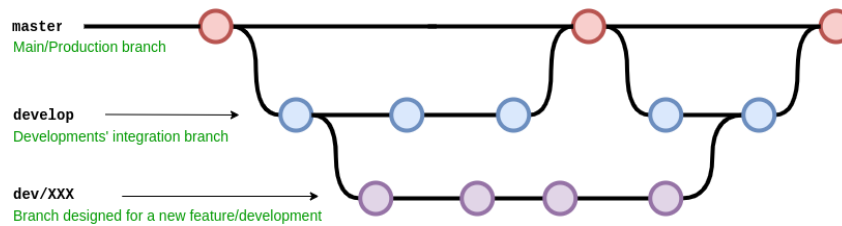


Fig. 55. Git flow

4.2 Docker images

As stated before, from the Docker standpoint, it usually requires a certain degree of specificity and manual steps to configure and build the Docker images. To overcome this, the makefiles recur to the pipreqs module and the Pack command-line interface (CLI) in order to streamline this whole process.

That is, as exemplified in Figure 56, pipreqs is used in order to generate the Python dependencies list (requirements.txt) according to the source code present in the given Python project, and the Pack CLI handled the build of the Docker images (based on the source code found and its list of dependencies).

```

build-docker-gateway:
@echo "${PATTERN_BEGIN} BUILDING GATEWAY PACK..."

@pipreqs --force --savepath requirements.txt.tmp
@sort -f requirements.txt.tmp > requirements.txt.tmp.sorted
@if cmp -s requirements.txt.tmp.sorted requirements.txt; then : \
else cp -f requirements.txt.tmp.sorted requirements.txt; fi
@rm -f requirements.txt.tmp
@rm -f requirements.txt.tmp.sorted

@pack build ${GATEWAY_PACK_NAME} \
-builder ${BUILDPACK_BUILDER} \
-pull-policy if-not-present \
-verbose

@echo "${PATTERN_END} GATEWAY PACK BUILT!"

```

determination
of the
Python
dependencies
needed

build of the
Docker image

Fig. 56. Build process example

Having the Docker image built and ready, the next step consists of running the Docker image (as addressed in Figure 57 as an example). In this step, several arguments are sent, namely the Docker network name (applicable in the simulator, gateway, and model containers), environment variables

(mostly but not only to indicate the RabbitMQ settings), the port in which the container will be published/run and the Docker volume name (used in the simulator, to point out the Docker volume used to store its data persistently).

```
start-docker-gateway
@echo "${PATTERN_BEGIN} STARTING GATEWAY PACK..."

@docker run -d \
  --name ${GATEWAY_CONTAINER_NAME} \
  --network ${SIMULATOR_NETWORK_NAME} \
  -e RABBIT_USER=${RABBIT_USER} \
  -e RABBIT_PASSWORD=${RABBIT_PASSWORD} \
  -e RABBIT_HOST=${RABBIT_CONTAINER_NAME} \
  -e RABBIT_MANAGEMENT_PORT=${RABBIT_MANAGEMENT_PORT} \
  -e RABBIT_PORT=${RABBIT_PORT} \
  -p ${GATEWAY_PORTS} \
  ${GATEWAY_PACK_NAME}

@echo "${PATTERN_END} GATEWAY PACK STARTED!"
```

(a) example - gateway

```
start-docker-simulator
@echo "${PATTERN_BEGIN} STARTING SIMULATOR PACK..."

@docker run -d \
  --rm \
  --name ${SIMULATOR_CONTAINER_NAME} \
  --network ${SIMULATOR_NETWORK_NAME} \
  --volume ${SIMULATOR_VOLUME} \
  -p ${SIMULATOR_FLASK_PORT_EXTERNAL} \
  -p ${SIMULATOR_WS_PORT_EXTERNAL} \
  -e SIMULATOR_HOST=${SIMULATOR_HOST} \
  -e SIMULATOR_WS_PORT=${SIMULATOR_WS_PORT} \
  -e FLASK_APP=simulator/main.py \
  ${SIMULATOR_PACK_NAME}

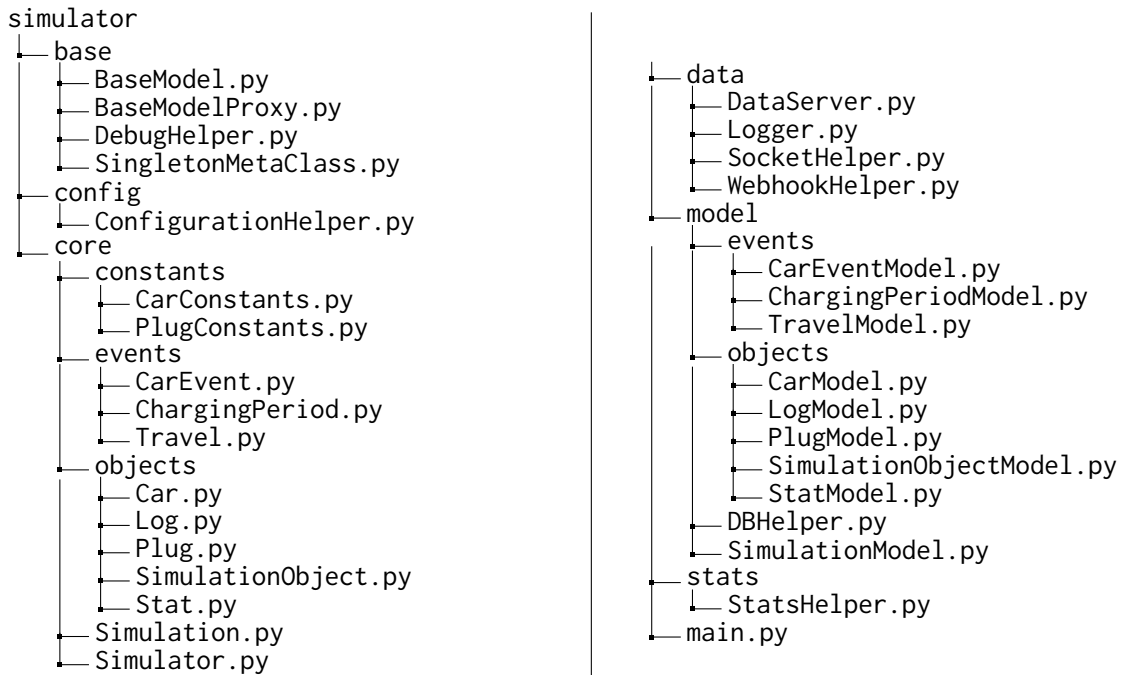
@echo "${PATTERN_END} SIMULATOR PACK STARTED!"
```

(b) example - simulator

Fig. 57. Run process examples

4.3 Simulator

The source files that encompass this component are organised as:



That is, the source code is divided into six different packages: **base** (in which there are several general utility and base classes), **config** (that contains a utility class that handles the simulator's configuration),

core (where the business logic and every object involved is implemented), **data** (with classes that operate the data processes revolving its exposure and communication), **model** (that includes the implementation of every ORM class and a utility class to handle the database communication and export processes) and **stats** (that encapsulates the preparation of the simulations' statistics).

4.3.1 Docker volume

Moreover, the simulator uses a Docker volume to store data persistently. To be precise, the volume used by the simulator stores persistently the **configuration file** and the **SQLite database file**.

Regarding the configuration file, it follows the structure shown in the following snippet.

```
{
  "number_of_cars": 10,
  "number_of_charging_plugs": 4,
  "sim_sampling_rate": 900000,
  "travel_affluence_multiplier": 1,
  "minutes_per_sim_step": 15,
  "number_of_steps": 96,
  "gateway_request_base_url": "http://cont_energysim_gateway:8000/{}",
  "enable_debug_mode": false,
  "webhook_url": "https://hooks.slack.com/services/XXXXXXXXXX"
}
```

This configuration is fundamental for this solution, since it provides the researchers an easy and flexible way to adapt the simulation process to their needs. Each possible configuration parameter is described in Table 9.

Configuration key	Description	Suggested value
number_of_cars	Number of cars per simulation	10
number_of_charging_plugs	Number of charging plugs per simulation	4
sim_sampling_rate	Sampling rate between each simulation step	900000
travel_affluence_multiplier	Travel affluence multiplier (1 = default, 0.5 = half the affluence)	1
minutes_per_sim_step	Number of minutes that each simulation step represents	15
number_of_steps	Number of steps per simulation	96
gateway_request_base_url	Gateway URL template	http://cont_energysim_gateway:8000/{}
enable_debug_mode	Enable/disable debug/verbose mode	false
webhook_url	Slack notification webhook URL	https://hooks.slack.com/services/XXXXXX

Table 9. Simulator configuration

The database consists of a SQLite .db file, as shown in Figure 58.

The image shows two screenshots of a SQLite database interface. The left screenshot displays the 'Estrutura do banco de dados' (Database structure) window, listing various tables such as 'Cars', 'ChargingPeriods', 'Logs', 'Plugs', 'SimulationStats', 'Simulations', 'Travels', and 'sqlite_sequence'. The right screenshot shows the 'Travels' table selected, displaying a grid of data with columns: 'id', 'car_id', 'start_datetime', 'end_datetime', 'distance', and 'tery_consumpt'. The data rows show simulation results for different cars and time periods.

	id	car_id	start_datetime	end_datetime	distance	tery_consumpt
1	1	31	2021-05-26 ...	2021-05-26 ...	12.0681934...	9.57751711...
2	2	32	2021-05-26 ...	2021-05-26 ...	14.2868995...	5.74027239...
3	3	32	2021-05-26 ...	2021-05-26 ...	7.85886144...	2.20427803...
4	4	32	2021-05-26 ...	2021-05-26 ...	8.41366767...	2.05544957...
5	5	33	2021-05-26 ...	2021-05-26 ...	12.1729583...	8.64638737...
6	6	34	2021-05-26 ...	2021-05-26 ...	20.9151897...	10.0
7	7	33	2021-05-26 ...	2021-05-26 ...	6.75654363...	4.81019078...

Fig. 58. SQLite database

4.3.2 Webhook

Regarding the Slack webhook, the simulator sends messages through it by calling the method `send_message` present in the `WebhookHelper` class. Technically speaking, it sends an HTTP request to the configured webhook URL, being that the request body is composed of the given message and its styling. An example of its usage is as follows in the following snippet.

```
( . . . )
def on_stop( self ):
    self.end_simulation( True )
    WebhookHelper.send_message( 'Simulation stopped!', 'INFO' )
( . . . )
```

Some examples of webhook messages can be found in Figure 59.

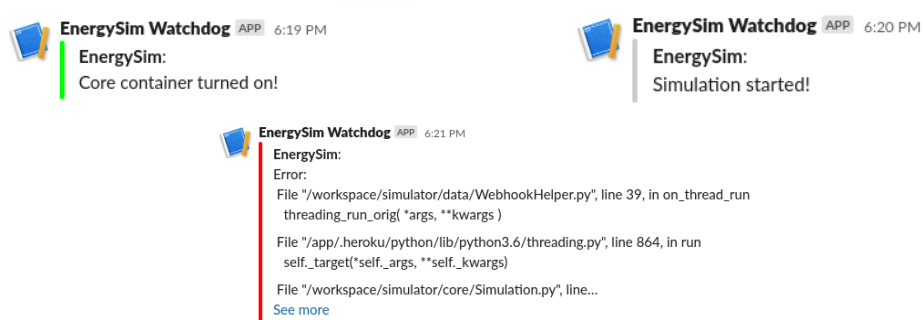


Fig. 59. Slack Webhook messages

4.3.3 ORM classes

As mentioned before, concerning the database operations, we recur to ORM classes that consume the SQLAlchemy library and do all these operations under the hood. The nature of these operations is based on how Python built-in objects work. In other words, a Python object instance represents a table row, and each Python object attribute represents a column value for a given table row. Therefore, we can create a new row by simply instantiating its class, and we can fetch or update its column values by plainly accessing or changing its attributes. Each model class is constructed similarly as the following example:

```
( . . . )
class LogModel( SimulationObjectModel ):

    csvFilename = 'Logs'
    class sqlmeta:
        table = 'Logs'

    _message = StringCol( default = '', dbName = 'message', title = 'message' )

    def get_message( self ):
        return self._message
```

```

    def set_message( self, message ):
        self._message = message
    ( . . . )

```

4.3.4 REST API

The REST API implemented in the proposed solution is exposed by the Flask framework. The implementation of its endpoints is located in the `DataServer` class. An example is shown in the following snippet.

```

( . . . )
@api.route( '/is_simulation_running' )
def is_simulation_running( ):
    simulator = DataServer.__simulator

    current_simulation = simulator.get_current_simulation( )

    response = None

    if current_simulation:

        is_simulation_running = current_simulation.is_simulation_running( )

        response = Response( json.dumps( { "is_simulation_running":
is_simulation_running } ), mimetype = 'application/json', status = 200 )

    else:

        response = Response( json.dumps( { "is_simulation_running" : False } ),
mimetype = 'application/json', status = 200 )

    return response
( . . . )

```

The implemented API endpoints are listed in Table 10.

4.3.5 WS messaging

As mentioned before, the proposed solution exposes the data through a WebSocket which can send and receive messages. That is, the simulator sends its state, its list of simulations alongside their data via WebSocket to the web clients and has the capability of receiving commands sent by the latter to trigger off a particular action.

Concerning the possible types of messages sent by the simulator, they are listed in Table 13 (present in the appendix).

Endpoint	Description
/plugs	Get the current simulation's list of plugs
/plugs/{plug_id}	Fetch info about a given plug of the current simulation
/plugs/{plug_id}/set_status/{new_status}	Set a new status for a given plug of the current simulation
/export	Export the entire database into .csv files
/is_simulation_running	Get the current simulation state
/start_new_sim	Start a new simulation
/get_sim_data_by_id/{simulation_id}	Fetch info about a given simulation

Table 10. REST API endpoints

All the possible WebSocket messages (or commands) that can be received and handled by the simulator are enumerated in Table 14.

4.4 Gateway

The Gateway is a microservice that serves as a broker for the simulator, unifying the communication between the simulator and the data models' microservices.

This is implemented recurring to the RabbitMQ framework, which provides a message-oriented middleware based on AMQP for the gateway and microservices' Docker containers. Thus, an instance of a RabbitMQ Docker image is executed alongside the simulator's gateway.

An appropriate Docker image is started for the gateway based on the RabbitMQ connection settings provided (host, port, username, and password). Then, in that same Docker image, the Nameko framework sets up and handles the connection between the container and the RabbitMQ message broker. Having the connection set up, the implementation class of the gateway service reaches the data models' containers through instances of `RpcProxy` (as illustrated in the following code snippet). These instances serve as bridges to the corresponding RPC client (based on a given name), being that, from the gateway standpoint, each RPC client represents a specific data model.

```
class GatewayService(object):
    ( . . . )
    name = 'gateway_energysim'

    rpc_model_travel_distance = RpcProxy( 'model_energysim_travel_distance' )
    rpc_model_charging_period_duration = RpcProxy( 'model_energysim_charging
    _period_duration' )
    rpc_model_charging_period_energy_spent = RpcProxy( 'model_energysim_charging
    _period_energy_spent' )
    rpc_model_travel_final_battery_level = RpcProxy( 'model_energysim_travel
```

```

_final_battery_level' )
rpc_model_travel_affluence = RpcProxy( 'model_energysim_travel
_affluence' )
rpc_model_travel_duration = RpcProxy( 'model_energysim_travel
_duration' )
( . . . )

```

With the proxies ready, the gateway will serve a Nameko microservice which will dispatch its requests to the appropriate data model through the respective `RpcProxy`. To dispatch the request to its corresponding data model, as shown in the following snippet, it is required to call the intended remote method through its `RpcProxy`. As a result, for every request made to the gateway, the request is dispatched to the respective data model, and its response is brought back to the gateway to be returned to the consumer.

```

class GatewayService(object):
    ( . . . )
    @http(
        "GET",
        "/travel/distance"
    )
    def get_travel_distance( self, request ):
        travel_distance = self.rpc_model_travel_distance.get_distance( )
        return Response(
            travel_distance,
            mimetype='application/json'
        )
    ( . . . )

```

The implemented gateway endpoints are enumerated in Table 11.

Endpoint	Description
/travel/distance	Generate a travel distance
/travel/final_battery_level/<initial_battery_level>/<travel_distance>	Generate the final battery level of a car after a travel (based on its initial battery level and the travel's distance)
/travel/affluence/<hour_of_day>	Generate a travel affluence (based on a given hour of the day)
/travel/duration	Generate a travel duration
/charging_period/duration	Generate a charging period duration
/charging_period/energy_spent/<charging_progress>	Generate a charging period energy expenditure (based on its progress)

Table 11. Gateway endpoints

4.5 Data models

Similar to the gateway, the data models also consist of microservices. In the same way as the gateway, their respective Docker images are started, and they connect to the same RabbitMQ message broker through the Nameko framework. With the connection set up, the data models are ready to be called up from the gateway.

As for the implementation class of each data model, it must name the implementing microservice to be referenced and called by the gateway (via `RpcProxy`). To name it, the class needs to have an attribute name in which we define it, as illustrated in the following snippet.

```
class ModelService:
    ( . . . )
    name = 'model_energysim_travel_distance'
    ( . . . )
```

Having the service named, it is needed to implement the remote method (that will be called from the gateway), which contains the business logic and returns its result to the gateway. In order to do that, in the implemented method, it is required to annotate it with the Nameko decorator `@rpc`, as exemplified in the following snippet.

```
class ModelService:
    ( . . . )
    @rpc
    def get_distance( self ):
        travel_distance = self.generate_distance( )
        response = json.dumps( { 'travel_distance': travel_distance } )
        return response
```

```
( . . . )
```

For the business logic revolving around data generation, we mostly used the TensorFlow framework in this thesis. More specifically, according to a given average value and a given standard deviation, TensorFlow was used to generate a random number. This is accomplished by utilizing its utility methods that, based on a particular minimum value and maximum value and a specific tensor, creates a TensorFlow session in which a random number is generated and gathered from a uniform distribution.

In the following code snippet, we can observe an example of its usage.

```
class ModelService:
    ( . . . )
    TRAVEL_DISTANCE_AVG = 12.421
    TRAVEL_DISTANCE_STDDEV = 8.967
    ( . . . )
    def generate_distance( self ):
        shape = [ 1,1 ]
        min_travel_distance = ModelService.TRAVEL_DISTANCE_AVG - ModelService.
            TRAVEL_DISTANCE_STDDEV
        max_travel_distance = ModelService.TRAVEL_DISTANCE_AVG + ModelService.
            TRAVEL_DISTANCE_STDDEV

        tf_random = tf.random_uniform(
            shape=shape,
            minval=min_travel_distance,
            maxval=max_travel_distance,
            dtype=tf.float32,
            seed=None,
            name=None
        )
        tf_var = tf.Variable( tf_random )

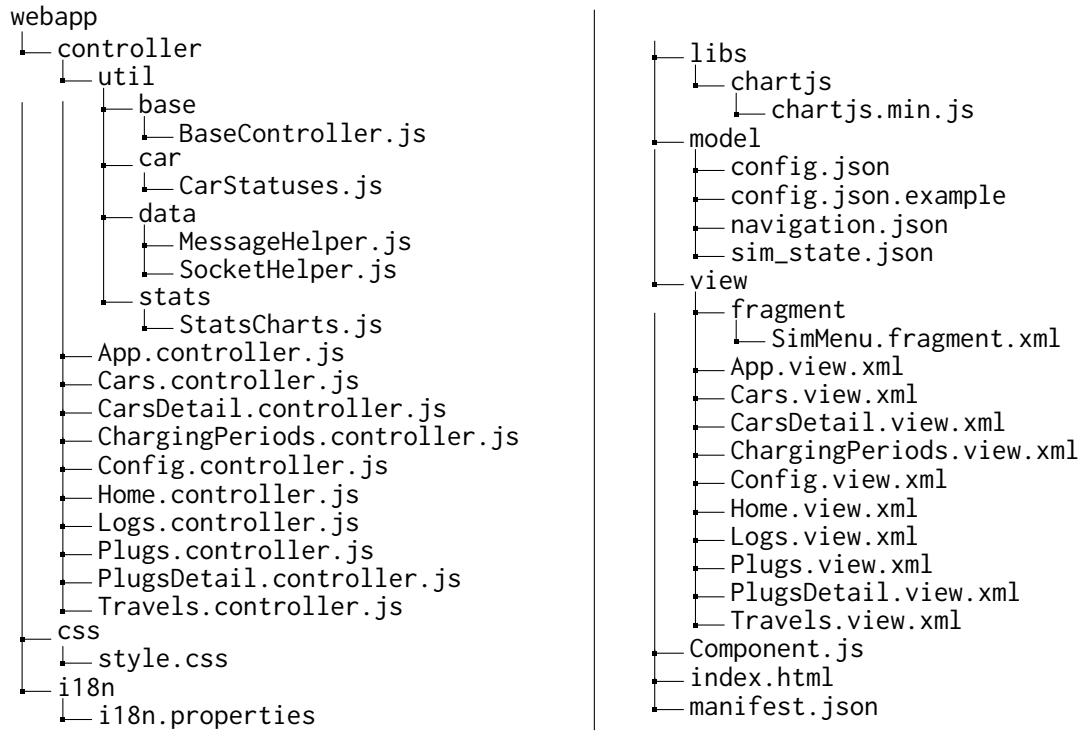
        tf_init = tf.global_variables_initializer( )
        tf_session = tf.Session( )
        tf_session.run( tf_init )

        tf_return = tf_session.run( tf_var )
        travel_distance = float( tf_return[ 0 ][ 0 ] )

        return travel_distance
    ( . . . )
```

4.6 Web client

Recurring to the OpenUI5 framework, the web client follows a MVC and plenty of open standards / developmental concepts, alongside a diverse range of usable UI controls. It followed the file structure drawn below:



In other words, it is divided in the following packages: **controller** (in which the view controllers are implemented), **css** (where the extra css stylesheets are located at), **i18n** (that contains the translatable text bundles), **libs** (that includes external libs, such as Chart.js), **model** (that accomodates all application models) and **view** (composed by all UI views and reusable fragments present in this web application).

4.6.1 Application manifest

The core of this web application is configured in the application manifest (the file named as `manifest.json`). Two of its key points consist of the configuration of the application's models and views.

Concerning the models, they can be configured similarly to the following snippet.

```

{
  ( . . . )
  "sap.ui5":
  {
    ( . . . )
  }
}
  
```

```

"models":
{
  ( . . . )
  "sim_data":
  {
    "type": "sap.ui.model.json.JSONModel"
  }
  ( . . . )
}
( . . . )
}
( . . . )
}

```

In regards to the process of constructing UI views, the application manifest requires the configuration of a route pointing to a target that is attached to a given view, as shown in the snippet below.

```

{
  ( . . . )
  "sap.ui5":
  {
    ( . . . )
    "routing":
    {
      ( . . . )
      "routes":
      [
        {
          "pattern": "Cars",
          "name": "Cars",
          "target": ["Cars"]
        }
      ],
      ( . . . )
      "targets":
      {
        ( . . . )
        "Cars":
        {
          "viewName": "Cars"
        }
      }
      ( . . . )
    }
  }
  ( . . . )
}

```

```

    }
    ( . . . )
  }
  ( . . . )
}

```

4.6.2 UI views

The OpenUI5 framework has several approaches regarding the development of UI views. The most prominent methodology is based on XML Views, consisting of .xml files in which the UI structure is defined and designed, similarly to the following XML snippet.

```

( . . . )
<mvc:View
  controllerName="com.perezjquim.energysim.client.controller.Config"
  xmlns="sap.m"
  xmlns:mvc="sap.ui.core.mvc"
  xmlns:code="sap.ui.codeeditor">
  <Page
    title="{i18n>Config}">
    <Label class="sapUiTinyMargin" text="{i18n>config_user}"/>
    <code:CodeEditor
( . . . )

```

Since the UI5 applications follow a MVC architecture, for each view, there is a corresponding controller. It can be observed in the following snippet.

<pre> (. . .) <mvc:View controllerName="com.perezjquim.energysim. client.controller.Config" xmlns="sap.m" xmlns:mvc="sap.ui.core.mvc" xmlns:code="sap.ui.codeeditor"> <Page title="{i18n>Config}"> <Label class="sapUiTinyMargin" text="{i18n>config_user}"/> <code:CodeEditor (. . .) </pre>	<pre> (. . .) sap.ui.define(["./util/base/BaseController", "./util/data/SocketHelper"], function(BaseController, SocketHelper) { "use strict"; return BaseController.extend("com.perezjquim.energysim.client .controller.Config", { onConfigSave: function(oEvent) { this.setBusy(true); (. . .) </pre>
--	--

Besides that, the same framework provides data-binding capabilities, being that data from the application models can be mapped directly to UI controls present on a given view. A suitable example is the

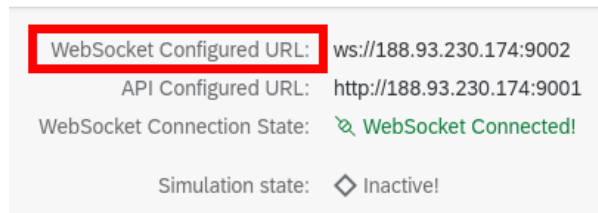


Fig. 60. i18n example

usage of translatable i18n texts (present in the `i18n.properties` file) in the application views. The required steps are illustrated in the following snippet.

<pre>(. . .) # > HOME VIEW ws_configured_url=WebSocket Configured URL ws_connection_state=WebSocket Connection State ws_is_connected=WebSocket Connected! ws_is_disconnected=WebSocket Disconnected! (. . .)</pre>	<pre>(. . .) <f:content> <Label text="{i18n>ws_configured_url}" /> <Text text="{config>/WS_URL}"/> (. . .)</pre>
---	--

Consequently, that text will be rendered as the label's text, as observed in Figure 60.

The UI events are handled via implementation of the UI control event callbacks. The usual procedure consists of naming in the XML view the controller method that will handle a given UI event. An example is shown in the following code snippet.

<pre>(. . .) <HBox justifyContent="SpaceBetween"> (. . .) <Button icon="sap-icon://excel-attachment" press="onPressExport" text="{i18n>export}" type="Emphasized" blocked="{= !(\${sim_state}/is_connected) && !\${sim_state}/is_sim_running)}"/> (. . .) </HBox> (. . .)</pre>	<pre>(. . .) onPressExport: function(oEvent) { this.setBusy(true); const oConfig = this.getModel("config"); const sAPIUrl = oConfig.getProperty("/API_URL"); const sExportUrl = `\${sAPIUrl}/export`; window.open(sExportUrl); this.setBusy(false); }, (. . .)</pre>
---	---

As a result, the implementation shown in the snippet above will result in the event handling shown in Figure 61.



Fig. 61. Event handling example

4.6.3 Configuration

The web client has a configuration of its own, expected to be in the `config.json` file. Thus, a file named as `config.json.example` is provided as a sample for this configuration. It can be described in Table 12 and exemplified in the snippet below.

Configuration key	Description	Suggested value
WS_URL	Simulator's WebSocket URL	ws://localhost:9002
API_URL	Simulator's REST API URL	http://localhost:9001
SAMPLE_SIM_CONFIG	Sample simulator configuration	<pre>{ "number_of_cars": 10, "number_of_charging_plugs": 4, "sim_sampling_rate": 900000, "travel_affluence_multiplier": 1, "minutes_per_sim_step": 15, "number_of_steps": 96, "gateway_request_base_url": "http://cont_energysim_gateway:8000/{}", "enable_debug_mode": false, "webhook_url": "https://hooks.slack.com/services/XXXX/XXXX/XXXX" }</pre>

Table 12. Web client configuration

{

```

"WS_URL": "ws://localhost:9002",
"API_URL": "http://localhost:9001",
"SAMPLE_SIM_CONFIG":
{
  "number_of_cars": 10,
  "number_of_charging_plugs": 4,
  "sim_sampling_rate": 900000,
  "travel_affluence_multiplier": 1,
  "minutes_per_sim_step": 15,
  "number_of_steps": 96,
  "gateway_request_base_url": "http://cont_energysim_gateway:8000/{}",
  "enable_debug_mode": false,
  "webhook_url": "https://hooks.slack.com/services/XXXX/XXXX/XXXX"
}
}

```

4.6.4 REST API

In the same way as the WebSocket, according to the configuration given to the web client, it connects to the simulator's REST API. It serves two purposes: for the **database export** (shown in Figure 61) and to browse through **data of previous simulations** (illustrated in Figure 62).

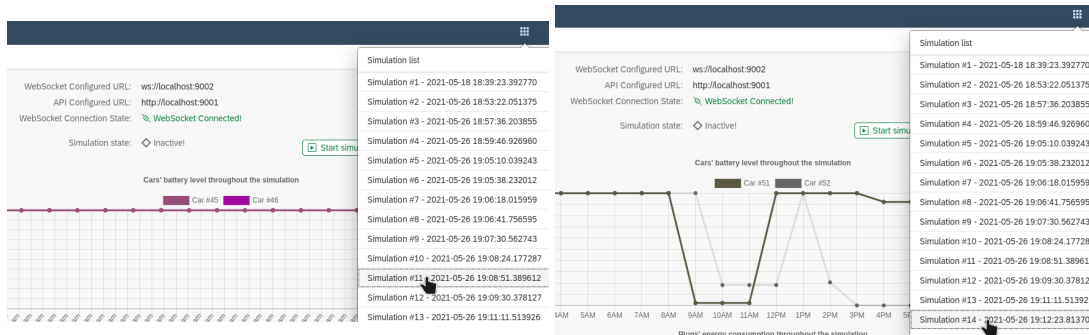


Fig. 62. REST API usage

4.6.5 WS messaging

The connection to the simulator's WebSocket is established recurring to a utility class that wraps around the calls to the WebSocket Web API. That is, based on the web client configuration, it connects to the intended WebSocket. Having the connection established, it can receive messages from the simulator (such as state and data changes) and trigger actions on the simulator (namely, the start of a new simulation). Some examples of WebSocket messages can be seen in Figure 63.

```

x Headers Messages Timing Initiator
All Enter regex, for example: (web)?socket
Data
{"message_type": "state", "message_value": {"is_sim_running": false, "config": {"number_of_cars": ...
{"message_type": "sim_list", "message_value": [{"id": 1, "is_running": null, "description": "Simulation...
{"message_type": "command", "message_value": {"command_name": "START-SIMULATION", "comma...
{"message_type": "state", "message_value": {"is_sim_running": true, "config": {"number_of_cars": 2, ...
{"message_type": "sim_list", "message_value": [{"id": 1, "is_running": false, "description": "Simulatio...
{"message_type": "data", "message_value": {"sim_datetime": "2021-06-15T00:00:00", "cars": [{"id": ...
{"message_type": "data", "message_value": {"sim_datetime": "2021-06-15T00:15:00", "cars": [{"id": ...
{"message_type": "data", "message_value": {"sim_datetime": "2021-06-15T00:30:00", "cars": [{"id": ...

```

Fig. 63. Example of WebSocket messages

4.7 Implementation final considerations

To recapitulate, the implementation of the solution was described above, from the Git flow and the building process of the Docker images to the composition of each component of the solution. Firstly, the general process behind the developments made follows a usual Git flow branching model. In the same way, the build process of the Docker images is addressed, consisting of Makefiles that generate the Docker images solely based on the source code found on the project in question (and its dependencies). Similarly, the simulator's implementation is also discussed, going through its file structure, the assembly of the SQLite database (and its operations), the webhook that communicates with Slack, the exposure of an external REST API, and its WebSocket communications. As for the gateway, it is a Nameko microservice that serves as a broker for the remaining microservices, dispatching the requests from the simulator to the respective data model through RPC proxies. The data models also consist of Nameko microservices, built and ready to be consumed by the gateway. Finally, the web client is composed of an OpenUI5 application that communicates with the simulator's WebSocket and external REST API and exposes its data on an enterprise-ready UI.

5 EVALUATION AND ANALYSIS

We went through three assessments to evaluate this solution: migrating this solution into a remote virtual machine, having an external developer enhance a given data model, and sharing this solution with a SMILE partner. This evaluation is crucial since it will help to identify if, when used in real-life scenarios, the implemented solution fulfills the objectives set in its proposal or not.

5.1 Migration of the solution

For the first assessment, access to a clean remote server (with Docker and Pack CLI pre-installed) was given, in which the implemented system would be deployed. The process behind the deployment consisted of cloning the Git repositories, configuring the simulator, and building the Docker containers. Due to the isolated nature revolving around Docker, the assessment went smoothly, and the solution was quickly migrated to the remote system, being hosted and widely accessible via web. However, two issues came up: a memory issue and a timeout issue. The first issue was noticed when, after a few weeks, the web clients could not establish a connection to the simulator's WebSocket, some simulations would get stuck, and there were some dumps/exceptions found on the simulator concerning the communication between the microservices. After a thorough analysis, it was noted that the remote virtual machine had an approximate 99% memory usage at the time. A large majority of this memory consumption was due to a machine learning process (unrelated to this simulator) running in the same machine that would consume more than half of the system's memory (about 70% of it). As a result, an adjustment had been made to limit the memory usage by the aforementioned memory-intensive process. Succeeding that same adjustment, the memory usage of the whole machine was stabilized and went to an average of 50%. With lessened memory usage, this problem never came up again. The second issue came up when, after running for a long time, the simulator would sporadically fail to reach its microservices, raising timeout errors. A research about the issue lead us to conclude that it was a common Docker issue when simultaneously running a large set of containers (as in this case). As indicated by several sources, the problem was solved by increasing the timeout values for the communications between the microservices. Since then, the solution has been up and running for 2 months with no issues.

5.2 Enhancement of a data model

Heading to the second assessment, we recurred to a Software Engineering degree graduate to test the installation/setup process and the easiness of addition/change of business logic into a particular data model. In order to do that, he was given documentation about the solution, its Git repositories, and its configuration, leaving him to perform the same procedure as in 5.1. Thus, he had prepared and installed the whole simulator environment to be able to test it on his own and implement it. The student found two setbacks in this process: a missing configuration file for the web client and a failing build process for the data models' containers. The first problem consisted of a missing configuration file (`config.json`) on the web client, making it unable to connect to the simulator's WebSocket and REST API. It was missing since it was git-ignored in the Git repository from which the student cloned the web client. Such files are git-ignored (in other words, not included in Git repositories) since they contain system-specific configurations. Therefore, a sample configuration file was introduced in the web client's repository - `config.json.example`. As a result, in the setup process, we have a sample configuration while git-ignoring the configuration file effectively consumed by the web client. The

second setback was caused by the fact that the student's system had a higher Python version installed locally (compared to the other systems in which the simulator was installed and set up), making the Pack CLI build the Docker images considering that exact version. However, the TensorFlow Python module was not available in that particular Python version. Consequently, the build process failed since it did not find the required dependency for the data models' Docker image. To overcome the issue, a `runtime.txt` file was included in the data models' Python projects with the one-lined content "python-3.6.12", being that the mentioned Python version supports TensorFlow. This new file is used to indicate the Pack CLI which runtime version of the environment (in this case, a Python one) is intended for the generated image. Having the runtime version specified, the build process could now successfully fetch the TensorFlow module and build those Docker images (independently of the local version of Python).

With the system set up and running, the student was given the task of developing a more sophisticated machine learning implementation on any data model present in the solution. More specifically, he was tasked to adapt the travel affluence's data model into a more advanced TensorFlow solution (as illustrated in Figure 64).

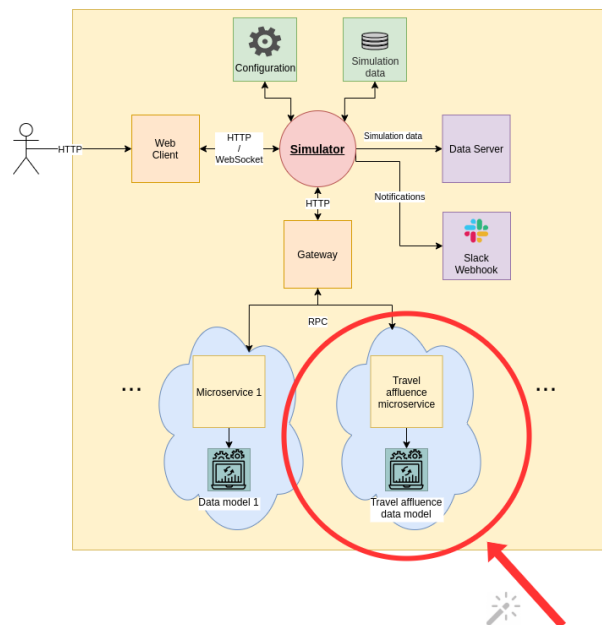


Fig. 64. Enhancement of a data model

Before any modification to the data model, he researched through several websites about data concerning the affluence of passengers on Funchal, resulting in the extraction of a pre-pandemic data sample of passengers coming from a given cruise during the several hours of the day.

With the gathered sample, the next phase consisted of the student transitioning the data model implementation from the hard-coded affluence values (based on the values gathered from SMILE

Project) into uniformly random generated values by TensorFlow (based on the new sample). The student quickly did this transition process.

Afterwards, the student was given the task of creating and implementing a self-learning TensorFlow model into the travel affluence's data model. Hence, the student followed an extensive and well-documented tutorial regarding the intended goal, which led him to build an example of a functional TensorFlow model successfully. However, due to the student's lack of time, the student could not build a model concerning the gathered sample and incorporate it into the travel affluence's data model.

5.3 Sharing with SMILE partners

Additionally, the proposed solution was shared with a SMILE partner - Trakm8¹⁶. This partner had originally access to real data gathered from another SMILE pilot. However, due to the lack of data brought by the Covid-19 pandemic, this solution will fill that gap and mitigate the lack of data. Therefore, the partner's route optimization planning production system¹⁷ would consume the REST API routes available in this simulator. So, having their smart-charging algorithms, the partner would then act upon the simulations and through the REST API. The tests revolving around this integration will be mainly fundamental to evaluate this solution's easiness of integration and not the efficiency of its smart-charging algorithms. As for the API routes, during this integration, they had a few minimal adjustments to fit the integration requirements of the partner's system. On 30th August 2021, those routes were successfully integrated and tested by the third-party mentioned earlier.

5.4 Conclusions

To summarize, we can conclude that this solution's evaluation results were very positive since it provided an easy installation/setup process. Most importantly, it provided an easy and agile environment that facilitated any modifications to it. The minor setbacks revolving around the setup were not impactful, yet they were quickly solved and helped in improving the solution. In terms of the modifications to the business logic, they were quickly made by an external Software Engineering degree graduate, even though not all objectives were achieved. Regarding the integration with the SMILE partner (Trakm8), the results showed that the solution fits their needs, and that their route optimization planning system is successfully consuming it to be used for commercial purposes. Focusing on the primary goal of this evaluation, the results indicate that the solution is easy to set up, easy to maintain/extend, and easy to integrate, as intended.

¹⁶<https://www.trakm8.com>

¹⁷<https://www.trakm8.com/route-optimisation-planning>

6 DISCUSSION

In this section, the related work and the proposed solution (alongside its implementation) will be recalled to compare the solution to the related work and to discuss the decisions made. This will help to identify the points taken from the related work (and how they were implemented) and whether the goals of the proposed solution were effectively fulfilled.

Recalling the proposed solution, it consisted of an open-source and easy-to-extend simulator of smart-charging algorithms, widely accessible through a Web client and with a flexible and decoupled architecture.

6.1 Development

Unlike existing tools such as PerMod[47], SAM[5], JANUS[6], V-Elph[7], SIMPLEV[10], ADVISOR[48] and MARVEL[24] (and similarly to SimSES[41]), this solution is built on Python, a more developer-friendly programming language that brings more code simplicity, readability and flexibility.

In the same way, in contrast to tools like PerMod[47], FreeGreenius[11], PSIM[33], JANUS[6], V-Elph[7], BLAST[31], SIMPLEV[10] and MARVEL[24] (and as adopted in SimSES[41] and ADVISOR[48]), this solution was made open-source and available to the public community.

6.2 Architecture

Besides that, based on SAM[5], PSIM[33], JANUS[6], V-Elph[7], and ADVISOR[48], this simulator adopts an open architecture and recurred to open design approaches, making it more favourable to future enhancements/modifications. The addition of microservices also accomplishes this, providing higher scalability to the solution, easing up on the deploy process, largely reducing downtime, making it simpler to maintain/extend, and enabling them to be programmed in any programming language. If the simulator needs another data model, the following is needed: creating the microservice and its implementation, including the respective RPC proxy in the Gateway, and then the corresponding HTTP call on the simulator. Consequently, this results in a decoupled and flexible system to maintain and develop on.

6.3 UI

By the same token as SimSES[41], SAM[5], FreeGreenius[11], PSIM[33], JANUS[6], V-Elph[7], BLAST[31], SIMPLEV[10] and ADVISOR[48] (while not present in PerMod[47] and MARVEL[24]), it also provides a visual representation of the simulated data, in order to provide an easy and hands-on way to analyze that same data.

As opposed to all the related work found, this solution included a web client. This client gives the researchers a solution within their reach, with no setup involved and no device specificity, bringing more practicality and convenience. In addition, with this web approach, the simulation results can be consumed by other researchers, even if not interested in the development of smart-charging algorithms. Moreover, the use of WebSockets allows the end-users to start/stop simulations, configure the simulator, and browse through the simulation data in real-time.

6.4 Deployment and integration

In terms of the goals of the proposed solution, the main ones were fulfilled. The implemented solution provided a versatile and user-friendly simulator for researchers to simulate smart-charging algorithms on. Moreover, according to the results analyzed in the previous section, this solution is abstract enough to be smoothly adjusted if intended, as proven by the fact that a Software Engineering graduate student was able to set it up and enhance it easily. In the same way, the flexibility of this solution is also proven by the fact that we were able to integrate this solution into a commercial service of a third-party partner associated with the SMILE project.

6.5 Conclusions

Concerning the decisions made, the selection of Python as the programming language was fruitful since it provided a productive and easier development, as expected. Furthermore, by recurring to a microservice-driven approach, it led to a scalable solution with an eased build/deploy process since it does not require the deployment of the entire system and has minimal impact/downtime. Moreover, the decision regarding the development of a web platform turned out to be a good one, considering it provided the researchers a user-friendly simulator within reach, regardless of the system and with no installation required. Additionally, the graphical representation of the simulation data was also a good addition because it favoured the data analysis that can be made on the simulator's web client by the researchers. Finally, the combination of the decisions mentioned above and the adoption of open design patterns and approaches facilitated further extensions/adaptations of the developed solution, as intended, fulfilling the primary goal of the proposal of this solution.

Summing up, it is implemented in Python, in the same way as SimSES[41], bringing simplicity and intuitiveness to the development phase. In addition, equivalently to SimSES[41] and ADVISOR[48], this solution was made open-source. Furthermore, based on the work done on SAM[5], PSIM[33], JANUS[6], V-Elph[7], and ADVISOR[48], its implementation recurred to an open architecture, a set of open design approaches and microservices. Hence, it improves the solution not only from the modifiability/extensibility standpoint, but also from the scalability, build and maintenance standpoint, resulting on a more decoupled and versatile system. Following the work done on the majority of the related work (SimSES[41], SAM[5], FreeGreenius[11], PSIM[33], JANUS[6], V-Elph[7], BLAST[31], SIMPLEV[10], and ADVISOR[48]), a graphic representation of the data was also provided, giving the researchers a visual and practical way to analyze its data. As an improvement over all the related work, this solution included a web client, giving the researchers an accessible and user-friendly web platform for them to simulate smart-charging algorithms without worrying about the processes revolving around its setup.

In a general way, we can observe that the work done in this solution was based on the majority of the related work, extending the good points from several existing solutions (such as SimSES[41], SAM[5], PSIM[33], JANUS[6]). The architectures observed in the related work pale in comparison to the implemented architecture in this solution. It recurs to more open design approaches and adopts the concept of microservices, making it a much more flexible, scalable, and maintainable solution than the existing ones. In the same way, the construction of a web platform for this solution surpasses the UIs present in the existing solutions. This is because it provides the researchers a hands-on tool with no

setup required and with state-of-the-art design/look-n-feel when compared to the desktop UIs given by the existing solutions.

7 CONCLUSION

To sum up, following a necessity of improving the charging process of EVs, the field of smart-charging and smart-charging algorithms emerged. However, the studies revolving around this field are complex, expensive, and risky. Consequently, this leads to a need for prior simulations in order to analyze/predict the integration of EVs in the electrical networks. Some simulators are already available, yet they consist of either academic, proprietary, or limited/rigid solutions. Therefore, in this thesis, we have presented a solution that provides a handy and intuitive tool for the researchers to simulate scenarios backed up by a simulation system with a decoupled architecture. Such architecture is materialized by adopting open design approaches and adopting the concept of containerized micro-services, easing up the process of maintaining/extending it and providing high scalability. When evaluating this solution's migration, enhancement and integration processes, it delivered good results, fulfilling its objectives and solving the stated problem. In addition, this solution is already up and running on a production system, while also being consumed externally by a SMILE partner.

In regards to the objectives of this thesis, they were all fulfilled, except the development of a travel route renderer, due to lack of time and also because it was a low-priority requirement without much payoff. Essentially, it consisted of a travel route renderer, with the capability of visually representing on a map the travels made during the simulations and their course. All the other contributions were fulfilled since the rest of the work regarding the literature review, the solution proposal, and its implementation were made.

The developments made are available in the following Git repositories:

- Simulator - <https://github.com/perezjquim/smartcharging-simulator-core>
- Web client - <https://github.com/perezjquim/smartcharging-simulator-client>
- Gateway - <https://github.com/perezjquim/smartcharging-simulator-gateway>
- Travel distance's data model - <https://github.com/perezjquim/smartcharging-simulator-service-travels-distance>
- Travel affluence's data model - <https://github.com/perezjquim/smartcharging-simulator-service-travels-affluence>
- Travel duration's data model - <https://github.com/perezjquim/smartcharging-simulator-service-travel-duration>
- Final battery level's data model - <https://github.com/perezjquim/smartcharging-simulator-service-travels-final-battery-level>
- Charging period duration's data model - <https://github.com/perezjquim/smartcharging-simulator-service-charging-period-duration>
- Charging period energy expenditure's data model - <https://github.com/perezjquim/smartcharging-simulator-service-charging-period-energy-spent>

On a personal note, one of the positive aspects of the work involved in this thesis is the fact that it got me doing an extensive literature review, which I have not done previously on an academic level. Besides that, another positive aspect is that this thesis brought me up to speed on several current software engineering and programming trends, as intended and expected. Some consisted of containerization (made with technologies like Docker), microservices, machine learning, and generally decoupled architectures. As a result, I became more familiar with such technologies/approaches, and I evolved as a software engineer.

7.1 Future work

As for the future, there is some work ahead to improve the proposed solution.

For instance, one of them is the development of the travel route renderer . More precisely, it should involve developing an additional data model and its integration with the simulator, with its gateway, and with its web client.

Another of them is the improvement/refinement of the data models to incorporate the full potential of TensorFlow models, utilizing fully self-learning models to meliorate the simulation process. By incorporating the full potential of TensorFlow, these data models can be advanced enough to be used for other purposes besides this simulator.

REFERENCES

- [1] 2019. *2019 Relatórios e Contas - Annual Report*. Technical Report. Empresa de Eletricidade da Madeira. https://www.eem.pt/media/733843/pt_eem_relatoriocontas_2019.pdf -.
- [2] 2020. European island imaginaries: Examining the actors, innovations, and renewable energy transitions of 8 islands. 101491. <https://doi.org/10.1016/j.erss.2020.101491>
- [3] AMQP. What is AMQP and why is it used in RabbitMQ? - CloudAMQP. <https://www.cloudamqp.com/blog/what-is-amqp-and-why-is-it-used-in-rabbitmq.html>
- [4] Maja Barring, Bjorn Johansson, Erik Flores-Garcia, Jessica Bruch, and Mats Wahlstrom. 2018. CHALLENGES OF DATA ACQUISITION MODELS FOR SIMULATION MODELS OF PRODUCTION SYSTEMS IN NEED OF STANDARDS. *IEEE*, 691–702. <https://doi.org/10.1109/WSC.2018.8632463>
- [5] Nate Blair, Nicholas DiOrto, Janine Freeman, Paul Gilman, Steven Janzou, Ty Neises, and Michael Wagner. *System Advisor Model (SAM) General Description (Version 2017.9.5)*. Technical Report. National Renewable Energy Laboratory. <https://www.nrel.gov/docs/fy18osti/70414.pdf>
- [6] J. R. Bumby, P. H. Clarke, and I. Forster. 1985. Computer modelling of the automotive energy requirements for internal combustion engine and battery electric-powered vehicles. 265–279. <https://doi.org/10.1049/ip-a-1.1985.0059>
- [7] K.L. Butler, M. Ehsani, and P. Kamath. 1999. A Matlab-based modeling and simulation package for electric and hybrid electric vehicle design. 1770–1778. <https://doi.org/10.1109/25.806769>
- [8] Lili Cao and John Krumm. 2009. From GPS traces to a routable road map (*GIS '09*). Association for Computing Machinery, 3–12. <https://doi.org/10.1145/1653771.1653776>
- [9] Chart.js. Chart.js | Open source HTML5 Charts for your website. <https://www.chartjs.org/>
- [10] G. H. Cole. 1991. *SIMPLEV: A simple electric vehicle simulation program, Version 1.0*. Technical Report DOE/ID-10293. EG and G Idaho, Inc., Idaho Falls, ID (United States). <https://doi.org/10.2172/10167537>
- [11] Jürgen Dersch and Simon Dieckmann. 2015. Techno-Economic Evaluation of Renewable Energy Projects using the Software greenius. 17–24. <https://doi.org/10.5383/ijtee.10.01.003>
- [12] Docker. Empowering App Development for Developers | Docker. <https://www.docker.com/>
- [13] Flask. Flask. <https://palletsprojects.com/p/flask/>
- [14] Holger Hesse, Michael Schimpe, Daniel Kucevic, and Andreas Jossen. 2017. Lithium-Ion Battery Storage for the Grid—A Review of Stationary Battery Storage System Design Tailored for Applications in Modern Power Grids. 2107. <https://doi.org/10.3390/en10122107>
- [15] Jordan Hoffmann, Yohai Bar-Sinai, Lisa M. Lee, Jovana Andrejevic, Shruti Mishra, Shmuel M. Rubinstein, and Chris H. Rycroft. 2019. Machine learning in a data-limited regime: Augmenting experiments with synthetic data uncovers order in crumpled sheets. eaa6792. <https://doi.org/10.1126/sciadv.aau6792>
- [16] Qilong Huang, Qing-Shan Jia, Zhifeng Qiu, Xiaohong Guan, and Geert Deconinck. 2015. Matching EV Charging Load With Uncertain Wind: A Simulation-Based Policy Improvement Approach. 1425–1433. <https://doi.org/10.1109/TSG.2014.2385711>
- [17] Vadim Kravcenko. pipreqs: Pip requirements.txt generator based on imports in project. <https://github.com/bndr/pipreqs>
- [18] James Larminie and John Lowry. 2003. *Electric vehicle technology explained*. J. Wiley, West Sussex, England ; Hoboken, N.J.
- [19] Gauthier Limpens, Stefano Moret, Hervé Jeanmart, and Francois Maréchal. 2019. EnergyScope TD: A novel open-source model for regional energy systems. 113729. <https://doi.org/10.1016/j.apenergy.2019.113729>
- [20] Make. Make - GNU Project - Free Software Foundation. <https://www.gnu.org/software/make/>
- [21] Hannah Mareike Marcinkowski. 2018. Reference energy simulation models for the three pilot islands (Samsø, Orkney, Madeira): Smart Island Energy Systems - H2020 Project SMILE Deliverable 8.1.
- [22] Hannah Mareike Marcinkowski. 2018. Short and medium-term scenarios for the three pilot islands (Samsø, Orkney, Madeira): Smart Island Energy Systems - H2020 Project SMILE Deliverable 8.2.
- [23] Charalampos Marmaras, Erotokritos Xydias, and Liana Cipcigan. 2017. Simulation of electric vehicle driver behaviour in road transport and electric power networks. 239–256. <https://doi.org/10.1016/j.trc.2017.05.004>
- [24] W. W. Marr and J. He. 1995. *MARVEL: A PC-based interactive software package for life-cycle evaluations of hybrid/electric vehicles*. Technical Report ANL/ES/CP-87322; CONF-9510282-1. Argonne National Lab., IL (United States). <https://www.osti.gov/biblio/184295>

- [25] Mário Martins. 2015. *Evaluation of energy and environmental impacts of electric vehicles in different countries*. Ph.D. Dissertation. Instituto Superior Técnico, Lisbon, Portugal. https://fenix.tecnico.ulisboa.pt/downloadFile/844820067125012/Thesis%20Mario%20Martins_Corrections%20final.pdf
- [26] A. S. Masoum, A. Abu-Siada, and S. Islam. 2011. 1–7. <https://doi.org/10.1109/ISGT-Asia.2011.6167125>
- [27] S. M. Mousavi G. and M. Nikdel. 2014. Various battery models for various simulation studies and applications. 477–485. https://econpapers.repec.org/article/eeerensus/v_3a32_3ay_3a2014_3ai_3ac_3ap_3a477-485.htm
- [28] Nicolai Müller, Stephanie Schenk, Patrick Hertzke, and Ting Wu. 2018. The global electric-vehicle market is amped up and on the rise.
- [29] Nameko. What is Nameko? – nameko 2.12.0 documentation. https://nameko.readthedocs.io/en/stable/what_is_nameko.html
- [30] Maik Naumann, Cong Nam Truong, Michael Schimpe, Daniel Kucevic, Andreas Jossen, and Holger C. Hesse. 2017. 1–6.
- [31] J. Neubauer. 2014. *Battery Lifetime Analysis and Simulation Tool (BLAST) Documentation*. Technical Report NREL/TP-5400-63246. National Renewable Energy Lab. (NREL), Golden, CO (United States). <https://doi.org/10.2172/1167066>
- [32] Ofgem. Implications to the transition to Electric Vehicles. <https://www.ofgem.gov.uk/ofgem-publications/136142>
- [33] S. Onoda and A. Emadi. 2004. PSIM-based modeling of automotive power systems: conventional, electric, and hybrid electric vehicles. 390–400. <https://doi.org/10.1109/TVT.2004.823500>
- [34] OpenUI5. OpenUI5. <https://openui5.org/>
- [35] Pack. Pack · Cloud Native Buildpacks. <https://buildpacks.io/docs/tools/pack/>
- [36] Richard Pump, Arne Koschel, and Volker Ahlers. 2019. https://www.thinkmind.org/download.php?articleid=service_computation_2019_1_20_18002
- [37] Ghanim Putrus, P. Suwanapingkarl, D. Johnston, E.C. Bentley, and Mahinsasa Narayana. 2009. <https://doi.org/10.1109/VPPC.2009.5289760>
- [38] Python. What is Python? Executive Summary. <https://www.python.org/doc/essays/blurb/>
- [39] RabbitMQ. Messaging that just works – RabbitMQ. <https://www.rabbitmq.com/>
- [40] Paul Rutter and James Keirstead. 2012. A brief history and the possible future of urban energy systems. 72–80. <https://doi.org/10.1016/j.enpol.2012.03.072>
- [41] SimSES. SimSES - EES. <https://www.ei.tum.de/ees/fp-ees/simses/>
- [42] SMILE. SMILE H2020. <https://www.h2020smile.eu/>
- [43] R. Spotnitz. 2003. Simulation of capacity fade in lithium-ion batteries. 72–80. [https://doi.org/10.1016/S0378-7753\(02\)00490-1](https://doi.org/10.1016/S0378-7753(02)00490-1)
- [44] SQLAlchemy. SQLAlchemy – SQLAlchemy 3.9.1 documentation. <http://www.sqlalchemy.org/>
- [45] TensorFlow. TensorFlow. <https://www.tensorflow.org/?hl=pt-br>
- [46] websockets. websockets – websockets 9.1 documentation. <https://websockets.readthedocs.io/en/stable/index.html>
- [47] Johannes Weniger, Tjarko Tjaden, Nico Orth, and Selina Maier. *Performance Simulation Model for PV-Battery Systems (PerMod)*. Technical Report. University of Applied Sciences Berlin (HTW Berlin). https://pv-speicher.htw-berlin.de/wp-content/uploads/PerMod_Documentation.pdf
- [48] K.B. Wipke, M.R. Cuddy, and S.D. Burch. 1999. ADVISOR 2.1: a user-friendly advanced powertrain simulation using a combined backward/forward approach. 1751–1761. <https://doi.org/10.1109/25.806767>
- [49] Zhile Yang, Kang Li, and Aoife Foley. 2015. Computational scheduling methods for integrating plug-in electric vehicles with power systems: A review. 396–416. <https://doi.org/10.1016/j.rser.2015.06.007>

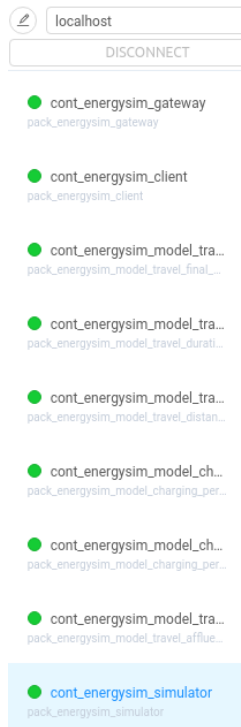


Fig. 65. Docker containers

A APPENDIX

Message type	Example	Description
state	<pre>{ 'message_type': 'state', 'message_value': { 'is_sim_running': 'XXX', 'config': { 'XXX' : 'XXX' } } }</pre>	Simulation state

data	<pre> { 'message_type': 'data', 'message_value': { 'sim_datetime': 'XXX', 'cars': [{ 'id': 'XXX', 'alias': 'XXX', 'simulation_id': 'XXX', 'status': 'XXX', 'travels': [{ 'id': 'XXX', 'car_id': 'XXX', 'car_alias': 'XXX', 'start_datetime': 'XXX', 'end_datetime': 'XXX', 'distance': 'XXX' 'battery_consumption': 'XXX' }] }], </pre>	Simulation data (date-time, cars, plugs, travels, charging periods, logs, and statistics)
------	---	---

```

    'charging_periods':
    [
      {
        'id': 'XXX',
        'car_id': 'XXX',
        'car_alias': 'XXX',
        'start_datetime': 'XXX',
        'end_datetime': 'XXX',
        'plug_id' : 'XXX',
        'plug_alias': 'XXX'
      }
    ],
    'battery_level': 'XXX',
    'plug_id': 'XXX',
    'plug_alias': 'XXX',
    'plug_consumption': 'XXX'
  },
  'travels':
  [
    {
      'id': 'XXX',
      'car_id': 'XXX',
      'car_alias': 'XXX',
      'start_datetime': 'XXX',
      'end_datetime': 'XXX',
      'distance': 'XXX'
      'battery_consumption': 'XXX'
    }
  ],
  'charging_periods':
  [
    {
      'id': 'XXX',
      'car_id': 'XXX',
      'car_alias': 'XXX',
      'start_datetime': 'XXX',
      'end_datetime': 'XXX',
      'plug_id' : 'XXX',
      'plug_alias': 'XXX'
    }
  ],
  'plugs':
  [
    {
      'id': 'XXX',
      'alias': 'XXX',
      'simulation_id': 'XXX',
      'status': 'XXX',
      'plugged_car_id': 'XXX',
      'plugged_car_alias': 'XXX',
      'energy_consumption': 'XXX',

```

```
'charging_periods':  
[  
  {  
    'id': 'XXX',  
    'car_id': 'XXX',  
    'car_alias': 'XXX',  
    'start_datetime': 'XXX',  
    'end_datetime': 'XXX',  
    'plug_id': 'XXX',  
    'plug_alias': 'XXX'  
  }  
]  
],  
'logs':  
[  
  {  
    'id': 'XXX',  
    'simulation_id': 'XXX',  
    'message': 'XXX'  
  }  
],  
'car_stats':  
{  
  'labels': [ 'XXX' ],  
  'datasets':  
  [  
    {  
      'label': 'XXX',  
      'backgroundColor': 'XXX',  
      'borderColor': 'XXX',  
      'fill': 'XXX',  
      'data': [ 'XXX' ]  
    }  
  ]  
},  
'plug_stats':  
{  
  'labels': [ 'XXX' ],  
  'datasets':  
  [  
    {  
      'label': 'XXX',  
      'backgroundColor': 'XXX',  
      'borderColor': 'XXX',  
      'fill': 'XXX',  
      'data': [ 'XXX' ]  
    }  
  ]  
},
```

	<pre> 'travel_stats': { 'labels': ['XXX'], 'datasets': [{ 'label': 'XXX', 'backgroundColor': 'XXX', 'borderColor': 'XXX', 'fill': 'XXX', 'data': ['XXX'] }] } </pre>	
sim_list	<pre> { 'message_type': 'sim_list', 'message_value': [{ 'id': 'XXX', 'is_running': 'XXX', 'description': 'XXX' }] } </pre>	List of simulations

Table 13. Types of WebSocket messages sent

Command type	Example	Description
START-SIMULATION	<pre>{ 'command_name' : 'START-SIMULATION' }</pre>	Start a new simulation
STOP-SIMULATION	<pre>{ 'command_name' : 'STOP-SIMULATION' }</pre>	Stop the current simulation
SET-PLUG-STATUS	<pre>{ 'command_name' : 'SET-PLUG-STATUS', 'command_args' : { 'plug_id' : 'XXX', 'new_status' : 'XXX' } }</pre>	Set a new status for a given plug
SET-CONFIG	<pre>{ 'command_name' : 'SET-CONFIG', 'command_args' : { 'new_config' : { 'XXX' : 'XXX' } } }</pre>	Update the simulator's config
SET-CONFIG-BY-KEY	<pre>{ 'command_name' : 'SET-CONFIG-BY-KEY', 'command_args' : { 'config_key' : 'XXX', 'config_value' : 'XXX' } }</pre>	Update a particular configuration of the simulator

Table 14. Types of WebSocket messages received