Summary

Formula Student is a world-wide engineering competition where students from different universities work to design, build, test and race a single-seater. ETSEIB Motorsport is the Formula Student team of ETSEIB that has taken part in this competition since 2007, developing year after year a new prototype.

In order to increase the performance of a car on track, the testing phase is key. During this phase, different parameters of the vehicle are modified in order to study its influence on the performance of the car and find the best range of values for each one to obtain the best performance during the competition.

This project aims to implement a tool to ease and standardize the testing procedure in the ETSEIB Motorsport team. The main goals are to find a solution that allows the team to register data following a standard procedure, that is device and location independent to guarantee access to all team members and, finally, that allows to export the registered data for further analysis with other software tools.

To meet the requirements, a web application has been developed using the Django Framework. The final output is a tool that meets all initial requirements and eases the procedure of data registration and consultation during the testing sessions with the prototypes developed by ETSEIB Motorsport.

ETSEIB

Index

Index of Figures

Index of Tables

Glossary

Event	Each of the different tests in which a Formula Student competition is divided.
Parameters	Configurable parts of the prototype that are object of study in the performance of the car.
Framework	In terms of Web, it is a collection of packages or modules which allow developers to write Web applications without having to handle such low-level details as protocols, sockets or process/thread management.
SOC	State of Charge. Percentage of energy available.
DRS	Drag Reduction System.
HTML	HyperText Markup Language
CSS	Cascade Style Sheets
JS	JavaScript
WSGI	Web Server Gateway Interface. Calling convention for web servers to forward requests to web applications or frameworks written in the Python programming language
RWD	Responsive Web Design. Approach in web design that renders web pages adapting to its size.
Werkzeug	Python WSGI utility library
Jinja2	Templating language for Python

1. Preface

1.1. Formula Student

Formula Student is a world-wide engineering competition where students have to design, build and race a single-seater. This competition has its origins in 1981 in the United States when the first event was organized by the *Society of Automotive Engineering* under the name of Formula SAE. Years later, in 1998, the first European competition took place in the United Kingdom organized by IMechE (The Institution of Mechanical Engineers) under the name of Formula Student.

Since this days, the competition has spread all around the globe holding competitions in Japan, Australia, different locations in Europe and America among others. This competition aims to approach students to real world problems and, as a consequence, narrow the gap between industry and university.

Teams can take part into two different categories: combustion vehicles or electric vehicles. Both categories take part in the same competition structure. It is based on different events where both the vehicle performance and the engineering skills of the teams are evaluated. Each competition can be divided into three parts:

- Scrutineering. Technical inspection of the vehicle. The aim is to determine whether the vehicle has been designed and built under the rules stablished by the competition. These rules have as purpose to guarantee the safety of the vehicle towards the driver and all the other attendants to the competition. It has no effect on the final score of the team, but it is necessary to succeed it to be able to take part in the dynamic events.

- Static events (325p). Three events with the aim to evaluate the engineering skills of the team.

    - Business Plan Event (75p). Students have to develop a business plan based on their built car and presented to an investors' committee which is represented by the judges.

    - Cost and Manufacturing Event (100p). The event consists on the development of a written report and a discussion with judges about the cost of the vehicle and its manufacturing process. Different documents are hand on where it is detailed the BOM of the prototype, the cost of required parts and alternative manufacturing processes to reduce the price or carbon footprint among other factors.

- Design Event (150p). The team has to present to judges - experts in the field - their prototype in terms of design. Design choices, evaluation of alternatives and the whole process followed to achieve the final solution are evaluated.

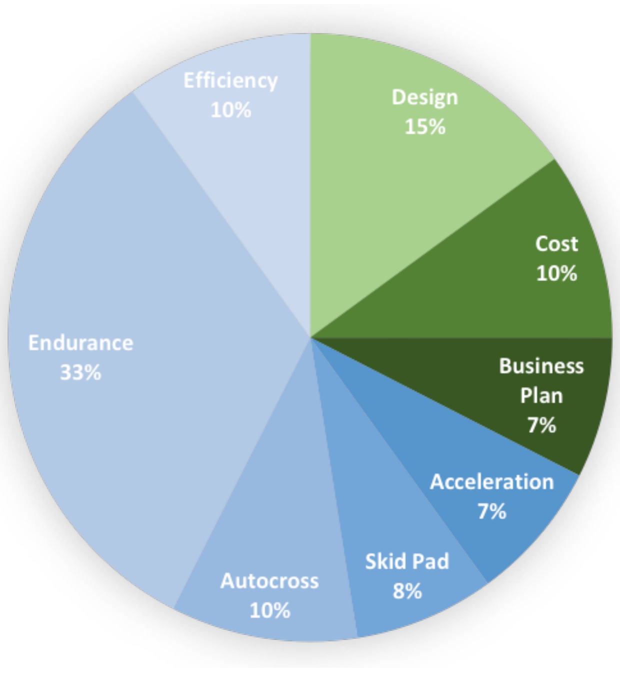


Figure. 1.1. Distribution of the score for a Formula Student event [1].

- Dynamic events (675p). The performance of the vehicle is evaluated on the track.
  - Acceleration (75p). The vehicle proves its acceleration capacity in a 75m straight.
  - Skid Pad (75p). Its aim is to prove the lateral acceleration capacity of the vehicle in an “eight-shape” circuit. The prototype has to perform two consecutive laps on each circle and only the second one is taken into account for the score.
  - Autocross (100p). The single-seater races on a circuit of around one-kilometre length through straights and curves. The time performed is taken into account as a reflex of the driving dynamics and handling qualities of the vehicle.

  - Endurance (325p). A 22km race proves the durability of the vehicle under long-term conditions. The race is completed by two drivers, each one performing 11km and with an intermediate stop to do a driver change.
    During the endurance, also the efficiency (100p) of the vehicle is evaluated as their fuel consumption or battery usage, depending on the vehicle type.

1.2. ETSEIB Motorsport

ETSEIB Motorsport is the Formula Student team of the *Escola Tècnica Superior d'Enginyeria Industrial de Barcelona*. This team has taken part in the competition since 2007. During the first four years, the team took part in the combustion category developing four different cars: from CAT01 to CAT04. In 2011 the team moved to the electric category developing its first single-seater: the CAT05e. Since that year ETSEIB Motorsport has developed a new vehicle year after year evolving through the time and obtaining great results in the different competitions.

Last year's season the team celebrated its tenth anniversary developing the CAT10e. This prototype is the sixth designed and built in the electric category. Its development structured the season in the following parts:

- Design. The design process implies defining the whole vehicle in terms of size, structure, materials, etc. A final CAD of the vehicle is achieved as a result.

- Build. The previously defined design becomes real. All the parts that make up the vehicle are manufactured and assembled together. The result is a ready-to-race prototype.

- Test. The vehicle needs to be tested and tuned before taking part in the competitions to ensure the best possible results are achieved. The time performed on the different events under different conditions is measured and studied to improve the performance of the car.

- Compete. In the season 2016-2017 ETSEIB Motorsport took part in the three most reputed competitions of electric Formula Student: Formula Student Austria (Red Bull Ring, July 31st – August 4th), Formula Student Germany (HockenheimRing, August 8th – August 13th) and Formula Student Spain (Circuit de Barcelona-Catalunya, August 23rd – August 27th).

1.3. Project origin

This year the team is developing the CAT11e, an electric single-seater that aims to beat the results of its predecessors as well as achieve the highest scores in all the competitions.

Part of the road to success is based on the testing period, when the car is tested on the racetrack to detect possible errors in the different systems as well as find the best configuration of parameters to achieve the maximum performance on the track. Is in this part of the development process where this project is framed.

The testing phase has gained importance in the season schedule during the last two years. The establishment of a standardized procedure and creation of a testing protocol turns into a necessity to avoid inefficient testing sessions and ensure the information transfer along time.

To study the evolution of the process until the date, only the two previous seasons will be taken into account. The main reason for this choice is the fact that there is no information available from previous years mainly because testing was carried along with competitions and done to prevent possible failures of the vehicle rather than tuning parameters and optimizing its performance.

Figure 1.1 CAT10e (season 2016-2017) during a testing session at Castellolí facilities

1.4. Motivations

The main motivation of this project is to improve the procedure of the testing process to achieve meaningful results in less time and ensure the access to this information along time for the development of future CATs.

Moreover, it must be taken into account also the motivation that generates the participation in a project such as Formula Student where all the previously acquired knowledge is proved on a real engineering problem.

Finally, this project aims to generate the base of a long-lasting tool that contributes to improve the quality of the work carried on in the upcoming seasons.

2. Introduction

2.1. Project Objectives

The aim of this project is to ease the data collection during the testing sessions as well as generate a standard procedure to facilitate the analysis of the evolution along the years. To achieve this main objective, the following requirements will have to be fulfilled during the development of this project:

- Study the actual procedure in data collection during the testing sessions, detect key weaknesses and possible solutions to them.
- Study of the needs with respect to the data that must be collected during the testing sessions.
- Develop a solution that matches the following requirements:
    - Ensure the possibility to register and save data in the required way for each event.
    - Guarantee easy access with different devices and different locations.
    - Possibility to consult and download the recorded data for further analysis with other software tools.
    - Generate an intuitive documentation to ensure its usage along the years.

2.2. Project Scope

This project found its framework in the season 2017-2018 of the team ETSEIB Motorsport. It will be developed to be tested before the current season’s competitions. Nevertheless, its usage for a concrete date cannot be guaranteed as it depends on the development of the CAT11e, the current season’s vehicle.

The collection of data will be done by hand and introduced by the user via the developed tool. The project will not deal with the collection of data via the telemetry system of the car neither the communication with any other data acquisition system implemented.

3. Background and viability studies.

During the design and manufacturing phases, the behaviour of the prototype is studied and simulated based on a model. Once the prototype is finished, this model has to be validated by reproducing the same challenges on the track. Moreover, the strengths and weaknesses of the car need to be found in order to optimize its performance on the competitions.

A normal testing session consist on the repetition of the same procedure (usually reproducing one of the events of the competition) changing different parameters and registering these changes. The aim is to study their influence on the performance and find the best possible combination to increase the possibility to achieve a higher score on the later competitions.

3.1. CAT09e

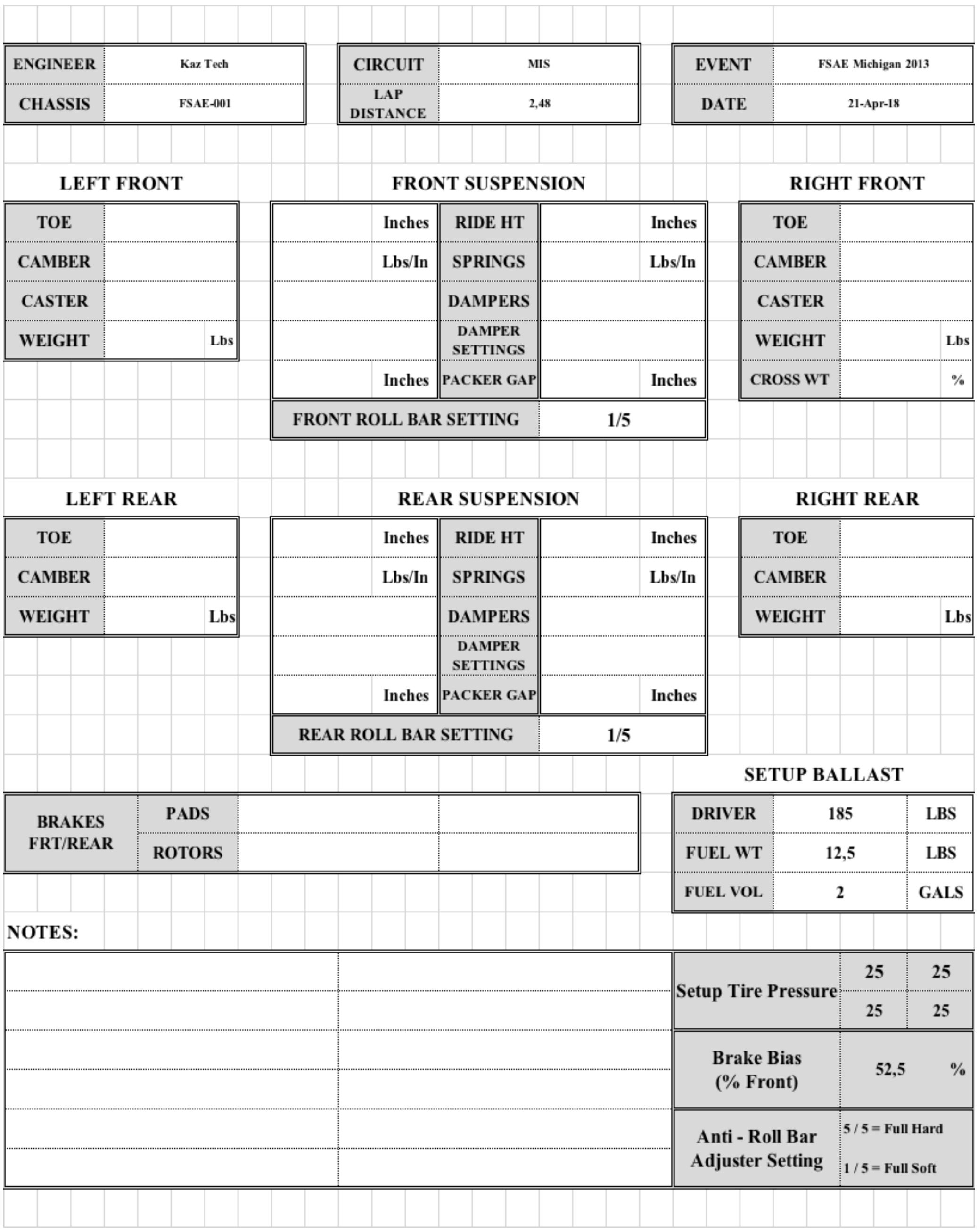
The CAT09e is the prototype developed in the season 2015-2016. This season was the first in which testing was carried out before competitions and integrated in the season schedule.

The registration of parameters was done by hand using templates. Those templates were designed by the dynamics department of the team and then, were mainly focused on the registration of dynamic parameters.

The dynamic parameters were registered together with the performance of the car and compared to the simulations based on the model of the vehicle. Other parameters related to powertrain and electronics were registered by hand too but not studied afterwards. Finally, the aerodynamic parameters were tested shortly (one testing session) to prove and validate simulation results.

As seen in *Figure 3.1,* the main parameters that were registered concerned the dynamics department. From this example, three different categories can be distinguished:

- **Environment:**
  - Responsible (Engineer)
  - Vehicle (Chassis)
  - Circuit
  - Lap Distance
  - Driver
  - Date
  - Fuel (SOC – Percentage of usable battery-of the battery in the case of an electric vehicle)
  - Event (in case the template is filled in during a competition)



ENGINEER	Kaz Tech	CIRCUIT	MIS	EVENT	FSAE Michigan 2013
CHASSIS	FSAE-001	LAP DISTANCE	2,48	DATE	21-Apr-18

LEFT FRONT

TOE		
CAMBER		
CASTER		
WEIGHT		Lbs

FRONT SUSPENSION

	Inches	RIDE HT		Inches
	Lbs/In	SPRINGS		Lbs/In
		DAMPERS		
		DAMPER SETTINGS		
	Inches	PACKER GAP		Inches
FRONT ROLL BAR SETTING			1/5	

RIGHT FRONT

TOE		
CAMBER		
CASTER		
WEIGHT		Lbs
CROSS WT		%

LEFT REAR

TOE		
CAMBER		
WEIGHT		Lbs

REAR SUSPENSION

	Inches	RIDE HT		Inches
	Lbs/In	SPRINGS		Lbs/In
		DAMPERS		
		DAMPER SETTINGS		
	Inches	PACKER GAP		Inches
REAR ROLL BAR SETTING			1/5	

RIGHT REAR

TOE		
CAMBER		
WEIGHT		Lbs

BRAKES FRT/REAR	PADS		
	ROTORS		

SETUP BALLAST

DRIVER	185	LBS
FUEL WT	12,5	LBS
FUEL VOL	2	GALS

NOTES:

Setup Tire Pressure	25	25
	25	25
Brake Bias (% Front)	52,5	%
Anti - Roll Bar Adjuster Setting	5 / 5 = Full Hard	
	1 / 5 = Full Soft	

Figure 3.1 Testing template during season 2015-2016. Registration of Setup parameters. The Setdown template register the same parameters but at the end of the session.

- **Suspensions' parameters:**
  - Toe
  - Camber
  - Caster
  - Weight

- Ride height
- Springs
- Dampers
- Damper setting
- Packer gap
- Roll bar

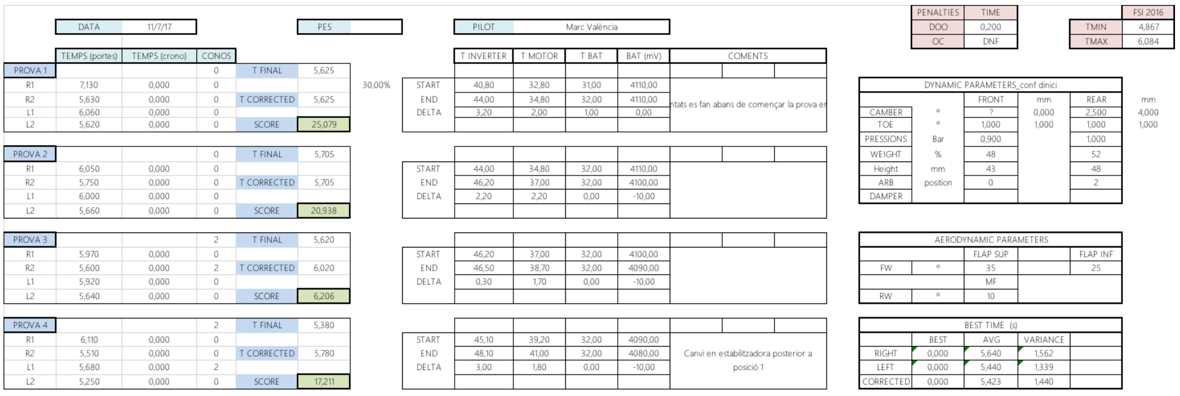
Each of these parameters, expect for the roll bar, is register for each wheel or axis of the prototype.

- **Other dynamic parameters:**
  - Brakes
  - Tire pressure
  - Brake bias
  - Anti-roll bar
  - Notes

3.2. CAT10e

Taking as base the previous season, the schedule of the development of the CAT10e integrated one month of testing previous of the competition.

In 2016-2017 season. The registration evolved from printed templates to templates in excel adapted to each event.



DATA	11/7/17	PES		PILOT	Marc València

PENALTIES	TIME
DOO	0,200
OC	DNF

	FSI 2016
TMIN	4,867
TMAX	6,084

	TEMPS (portes)	TEMPS (crono)	CONOS		
PROVA 1			0	T FINAL	5,625
R1	7,130	0,000	0		
R2	5,630	0,000	0	T CORRECTED	5,625
L1	6,060	0,000	0		
L2	5,620	0,000	0	SCORE	25,079
PROVA 2			0	T FINAL	5,705
R1	6,050	0,000	0		
R2	5,750	0,000	0	T CORRECTED	5,705
L1	6,000	0,000	0		
L2	5,660	0,000	0	SCORE	20,938
PROVA 3			2	T FINAL	5,620
R1	5,970	0,000	0		
R2	5,600	0,000	2	T CORRECTED	6,020
L1	5,920	0,000	0		
L2	5,640	0,000	0	SCORE	6,206
PROVA 4			2	T FINAL	5,380
R1	6,110	0,000	0		
R2	5,510	0,000	0	T CORRECTED	5,780
L1	5,680	0,000	2		
L2	5,250	0,000	0	SCORE	17,211

30,00%

	T INVERTER	T MOTOR	T BAT	BAT (mV)	COMENTS
START	40,80	32,80	31,00	4110,00	ntats es fan abans de començar la prova er
END	44,00	34,80	32,00	4110,00	
DELTA	3,20	2,00	1,00	0,00	
START	44,00	34,80	32,00	4110,00	
END	46,20	37,00	32,00	4100,00	
DELTA	2,20	2,20	0,00	-10,00	
START	46,20	37,00	32,00	4100,00	
END	46,50	38,70	32,00	4090,00	
DELTA	0,30	1,70	0,00	-10,00	
START	45,10	39,20	32,00	4090,00	Canvi en estabilitzadora posterior a posició 1
END	48,10	41,00	32,00	4080,00	
DELTA	3,00	1,80	0,00	-10,00	

DYNAMIC PARAMETERS_conf dinici					
		FRONT	mm	REAR	mm
CAMBER	º	?	0,000	2,500	4,000
TOE	º	1,000	1,000	1,000	1,000
PRESSIONS	Bar	0,900		1,000	
WEIGHT	%	48		52	
Height	mm	43		48	
ARB	position	0		2	
DAMPER					

AERODYNAMIC PARAMETERS				
		FLAP SUP		FLAP INF
FW	º	35		25
		MF		
RW	º	10		

BEST TIME (s)				
	BEST	AVG	VARIANCE	
RIGHT	0,000	5,640	1,562	
LEFT	0,000	5,440	1,339	
CORRECTED	0,000	5,423	1,440	

Figure 3.2 Template for Skid Pad event used in season 2016-2017 by ETSEIB Motorsport in the development of the CAT10e.

The *Figure 3.2* shows one example of template. Each template was adapted to the specific event carried on and was slightly modified during the session to adapt better to the needs and circumstances of the day.

The main changes introduced with respect to the previous season are:

- Creation of a specific template for each event.
- Registration of parameters concerning different sections (dynamics, powertrain and aerodynamics) in the same sheet.
- Registration of both setup and setdown in the same sheet.

In the previous example four parts can be distinguished:

- **Left column group.** Registration of the performance of the vehicle in each "run". The performance is measured as the time required to complete each lap and the number or cones touched during the run.
- **Middle column group**. Registration of the effect on different systems of the car. Specifically, it takes record of the effect of each run in the temperature of the motor, inverter and battery and the state of charge of the battery. The effect is measured as the difference in the previous mentioned values between the beginning and end of the run.
  It also includes a final space for comments. Changes of some parameters are registered in this section.
- **Right column group.** Registration of all dynamic, aerodynamic and powertrain initial parameters.
- **Other relevant information** is registered in the upper part of the template. This information is the date, driver and its weight. Although the initial plan was to keep track of the driver's weight during the season this parameter was hardly ever registered.

3.3. Future requirements

Based on the experience of the two previous seasons, the requirements of the tool to be developed are listed below:

- Flexibility – different templates for different events
- Registration of setup and setdown
- Registration of the performance in each run and possible problems (comments/notes section)
- Registration of characteristics not concerning the vehicle directly:
  - Circuit
  - Driver
  - Date

- Prototype parameters:
  - Dynamic parameters
    - Camber
    - Toe
    - Ride height
    - Brakes temperature
    - Tire's pressure
    - Weight
    - Dampers' setting
    - Roll and antiroll bar
  - Powertrain parameters
    - SOC
    - Battery's temperature
    - Motor's temperature
    - Inverter's temperature
    - Mode (sets the current limitation and other electric parameters)
  - Aerodynamic parameters
    - Adjustable wings' configuration
    - DRS availability

3.4. Final Functionalities.

The two main functionalities of the presented solution are explained in sections 3.4.1 and 3.4.2 as from a user point of view. Technical details about the implementation are developed in section 5.

3.4.1. Registration of a testing session.

When performing a testing session, the following information needs to be registered:

- Information concerning the environment (driver, location, day)
- Information about the car status (setup)
- Information about the car performance (the performance is measured as the time needed to complete a run)
- Information about the impact on the car (influence of a run in the different systems of the prototype).

This information cannot be registered in a random order but only in the previously mentioned one as otherwise the relation between the different fields could not be the expected one. That will result in a bad analysis of the data.

Therefore, the process will be set to be the following:

1. Selection of the event to be performed.
2. Registration of data concerning the environment.
    a. The driver is chosen from a driver's database. If the required driver has not been previously registered, the user will have to first create its profile before continuing with the event registration.
3. Registration of setup of the car. Initial parameters concerning the different systems of the vehicle.
4. Registration of runs:
    a. Before the run, registration of the state of those systems whose changes is subject of study.
    b. After the run, registration of the time and state of the previously mentioned systems.
5. While the setup of the car is not changed, registration of runs can be performed. If any parameter needs to be changed, it will be mandatory to go back to point 2 and make the concerning changes in the previously registered setup.

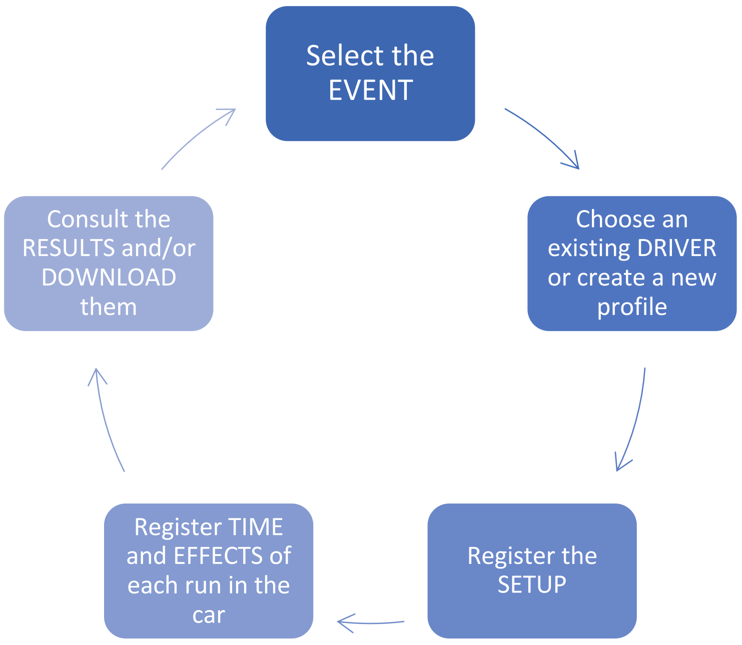

Select the EVENT
Choose an existing DRIVER or create a new profile
Register the SETUP
Register TIME and EFFECTS of each run in the car
Consult the RESULTS and/or DOWNLOAD them

Figure 3.3 Process to create a new testing session.

3.4.2. Request old data

At section 3.4.1, it is shown how to register data and the procedure followed for each testing session. But as important as registering these data is to be able to analyse it. Therefore, it is important to have access to the previously registered data.

In the developed tool, two different ways to access the information have been developed:

- Online access. It allows to consult the data registered up to the moment by accessing it through the different events. The information is shown in a table where each run corresponds to a row. The columns are different fields concerning the event (data, location, driver, lap time, setup identifier, etc.) but in a reduced version to guarantee it can be easily read by the user.
  Apart from the events' information, the user has access to the list of drivers registered up to the moment and all the used setups in the different sessions. That is useful as all runs are associated with a driver and a setup and therefore, further information of both the driver and the setup can be consulted in more detail by accessing to the correspondent table.

- File access. As important to be able to consult data is to be able to analyse it. Analysis of data recorded by different sensors in the vehicle is performed with third party software after the testing sessions. Being able to integrate the data recorded by the prototype's sensors with the one registered via the WebApp becomes a key feature for the utility of the developed tool.

With this aim, all the information is available to be downloaded as .csv file. A different format is given to the files depending on the information requested but, as in the online consultation, each row represents one entry of the table while the columns refer to different parameters concerning the topic.

4. Study of possible solutions.

As mentioned previously in section 2.1, the purpose is to develop a tool that eases the registration of information during testing sessions and its further analysis. Up to now, paper templates or excel sheets have been used providing good results but being always constrained by the further availability of the information.

The option of commercial software solutions has been discarded due to the high degree of customisation required for the type of tests developed.

Finally, to cope with the previously mentioned limitations, the idea of developing a WebApp tool has raised as a way to provide an easy-usable and accessible solution as well as ensure the correct storage of the information without limiting its access.

The main reasons leading to this choice are the availability of a multidevice tool allowing both consultancy and registration of data independently of the location and the owner of the device. Moreover, it allows flexibility in terms of possible future improvements while still creates a standardized procedure to ease data comparison along the seasons.

4.1. Study of the tools for the implementation of a WebApp

A study of the available tools in the market for the implementation of the solution is conducted in the following section dividing its content of the different needs.

4.1.1. Front-end

Concerning the front-end of the site to be developed, the main requirements are functionality and adaptability without neglecting the importance of the aesthetics. The tool must be intuitive and user-friendly, it means, easy to use.

For its development, the three basic tools to be used are:

- Hypertext Markup Language (**HTML**). Defines and structures the content of the website by using a simple markup language.
  The version to be used is HTML5, the latest version of HTML released on October 2014 [2]. This markup languages will allow to define the structure and content of the site.
- Cascade Style Sheets (**CSS**). Controls the styling (font, layout, etc.) of the site.

- JavaScript (**JS**). Language that allows the control of the behaviour of different elements. This programming language will be used to implement functions to make the site dynamic and allow the interaction with the content.

These three tools are the basics for the development of a website. For their implementation, a basic knowledge must be first acquired what will give the implementation of this project an educative approach too.

In addition to the previously mentioned tools, **Bootstrap 4** library will be used. It is an open-source front-end component library for developing responsive sites with HTML, CSS and JS [4]. It provides templates for different front-end objects easing their implementation and avoiding code repetition along the site.

4.1.2. Framework

For the development of a web application, different software frameworks are available. They provide a standard way to build web applications by automatizing common activities performed in web development.

Among all the possibilities, only those frameworks implemented in Python will be considered. Python is a high-level programming language for general-purpose. It is also the language taught at ETSEIB, one of the reasons why it has been chosen for the framework as it does not require learning a new programming language. Moreover, Python is currently one of the most used programming languages for data analysis what results in a great amount of information for learning to implement specific functions as well as a wide variety of public libraries concerning this topic.

The three possible frameworks considered has been Flask, Pyramid and Django.

- **Flask**: microframework written in Python based on *Werkzeug* and *Jinja 2.* It does not include database abstraction layer, form validation or any other functionality that a pre-existing third-party provide.
- **Pyramid**: minimalistic platform-independent framework written in Python and based on WSGI. It may be a really flexible framework what, in this case, may result into a difficulty as makes the learning curve harder at the beginning [3].
- **Django**: high-level Python Web framework that follows the model-view-controller (MVC) architectural pattern. Its main goal is to ease the creation of database-driven websites.

The final decision has been to adopt Django as a framework due to its easy and quick implementation. Although Django is initially though for bigger projects and other frameworks such as Flask will suit better a project of the size of the one to be implemented, Django has two main advantages in favour:

- Admin Panel. It provides an automatic and customizable admin panel that eases the management of the content.
- Database Management. It includes an ORM (Object-Relational Mapping) that allows relational databases to interact with the data generated in the application. Django allows creating models, tables and forms that ease the handling of data even without previous knowledge of databases.

4.2. Django structure and basic concepts.

Django is a python framework based on an MVC architecture. MVC stands for Model-View-Controller which is a software design pattern for developing web applications. The Model is the part responsible for handling the data (it deals with the database). The View is responsible for the display of the information requested by the user. Finally, the Controller is the part controlling the interaction between the Model and the View.

In the particular case of Django, the architecture is set as MTV (Model – Template – View). This is a special case of MVC. The main difference is that Django itself takes care of the controller part while the developer has to take care of the Templates. Templates are the representation layer containing information about the display of the data.

Django's architecture can be further explained but this topic will not be addressed in detail. Only relevant information to understand the structure of the developed solution and how it works will be first explained. For further information about Django architecture or steps to implement a project in Django, a project to be consulted is:

- Espais ETSEIB. Desenvolupament d'una pàgina web amb un framework de Python (Sun Xu, Wan Li; 2015) [8]

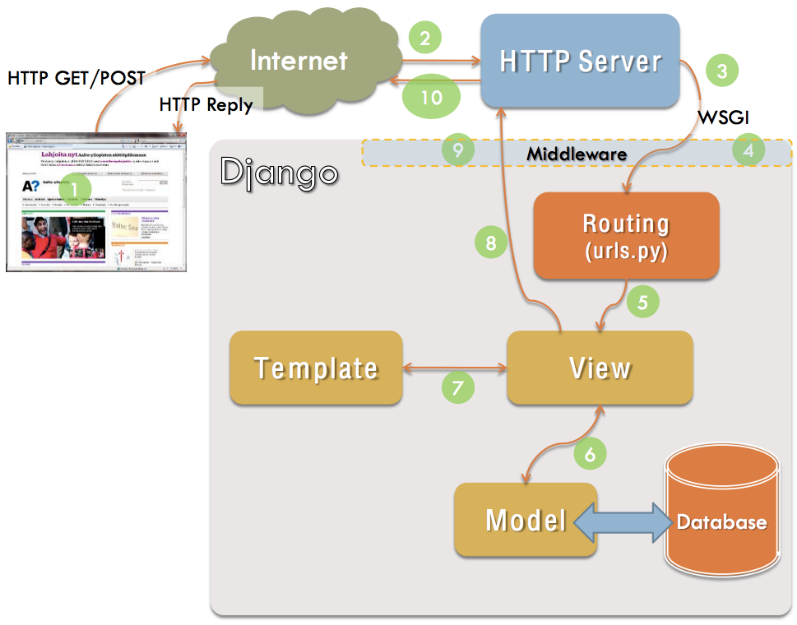


Figure 4.1 Django architecture. Visual representation of the interaction between the different components of a Django project whenever a user is interaction with the web. Retrieved from: Devopedia. 2017. "Django." Version 17, November 1. Accessed 2018-05-19. https://devopedia.org/django

4.2.1. Django Templates, Blocks and Variables.

A Django Template is a text file that can generate any text-based format such as HTML. Django templates allow to define the user-facing layer separating the design from the logic of the application.

Django Templates allow defining variables inside the content to be displayed. They are placed between curly brackets {{ }} and whenever the template engine encounters a variable, it will evaluate it and replace it with the result. The variables are passed to the template through the context. The context has the form of a Python dictionary where the keys are the variables and the arguments are the value of these variables.

Django Templates also incorporate an inheritance system that allows sharing a HTML code through different templates that have common parts. The system allows to insert the not shared parts as blocks in the template. Those blocks are also text files that define the view of a context.

The usage of templates, variables and blocks in the project is explained in section 5.3.

4.2.2. Django Models and Forms.

A Model is a description of the data stored in the database. It contains the fields and behaviors of the data being stored.

Models are defined as Python classes that inherit from a the *models.Model* Django class which is the one handling many of the features that Django classes offer and the interaction with the database.

Each model corresponds to a table in the database (unless otherwise specified) and each instance of the corresponding class is related to a row in the table. Fields correspond to columns, and they can be specified as Django Fields. Django Fields provide useful features such as information verification. It means that if a field is specified to be an email field and the user enters a birthdate, Django will raise up an error asking the user to verify the provided data as it appears to be incorrect.

Additionally, each Django Model can be associated to a Form that will allow to create an HTML view of a form with the same fields that has the model to which is related (unless specific modification). Again, data can be validated thanks to specifying the type of field in the model. Thanks to widgets, the display of the form can be handled by the designer of the page alternatively from the default display provided by Django.

Finally, another characteristic to point out are the relational fields. Those are *ManytoManyField*, *OnetoOneField* and *ForeignKey.* Those fields allow to designate a model instance as an attribute of another model instance. Relational fields and their usage in the project are further explained in section 5.4.1, specifically trough the Skid Pad model.

4.2.3. Django Views.

Django projects always contains a views.py file which is the layer in charge of handling the requests of the page.

The views file contains functions or classes that matches requests from the user with a specific content. Each view is responsible of either returning an *HttpResponse* object containing the content to be displayed or raising an exception. Its basic function is to retrieve data based on the parameters obtained, load a template and render it including the retrieved data.

Each request has a special attribute that distinguish it. It can be either GET or POST type which will define the type of interaction between the user and the database. GET requests are used when only to consult data and when there is no further interaction between the user and the database. Instead, POST requests are used when the user is sharing information to be included in the database or processed in another way.

Most part of the views used in this project are defined as classes that inherit from *Django.views.generic.TemplateView* superclass. Those classes incorporate methods to treat in a different way GET or POST requests.

5. Implementation.

The current project has been implemented following an iterative process. First, a basic approach has been reached to ensure the requirements were met even in a rudimentary way. From that basic structure, more complex relations have been introduced to make the user-experience better and the quality of the recorded data of higher grade.

As a result, the application developed meets all the initial requirements and offer some extra ones adopted under suggestion of the final users, team members of ETSEIB Motorsport.

5.1. Solution schema.

The structure of the solution has been designed following the information flow described in Registration of a testing session.3.4.1

Figure 5.1 represents the structure chosen. It explains how the user moves along the page starting from the *Home* page to any point in the application following the logical path. There is also the possibility to move from point A to B (A and B being any page of the application) using the navigation bar.

The points marked with the database icon represent a process where a new instance of a model is created.

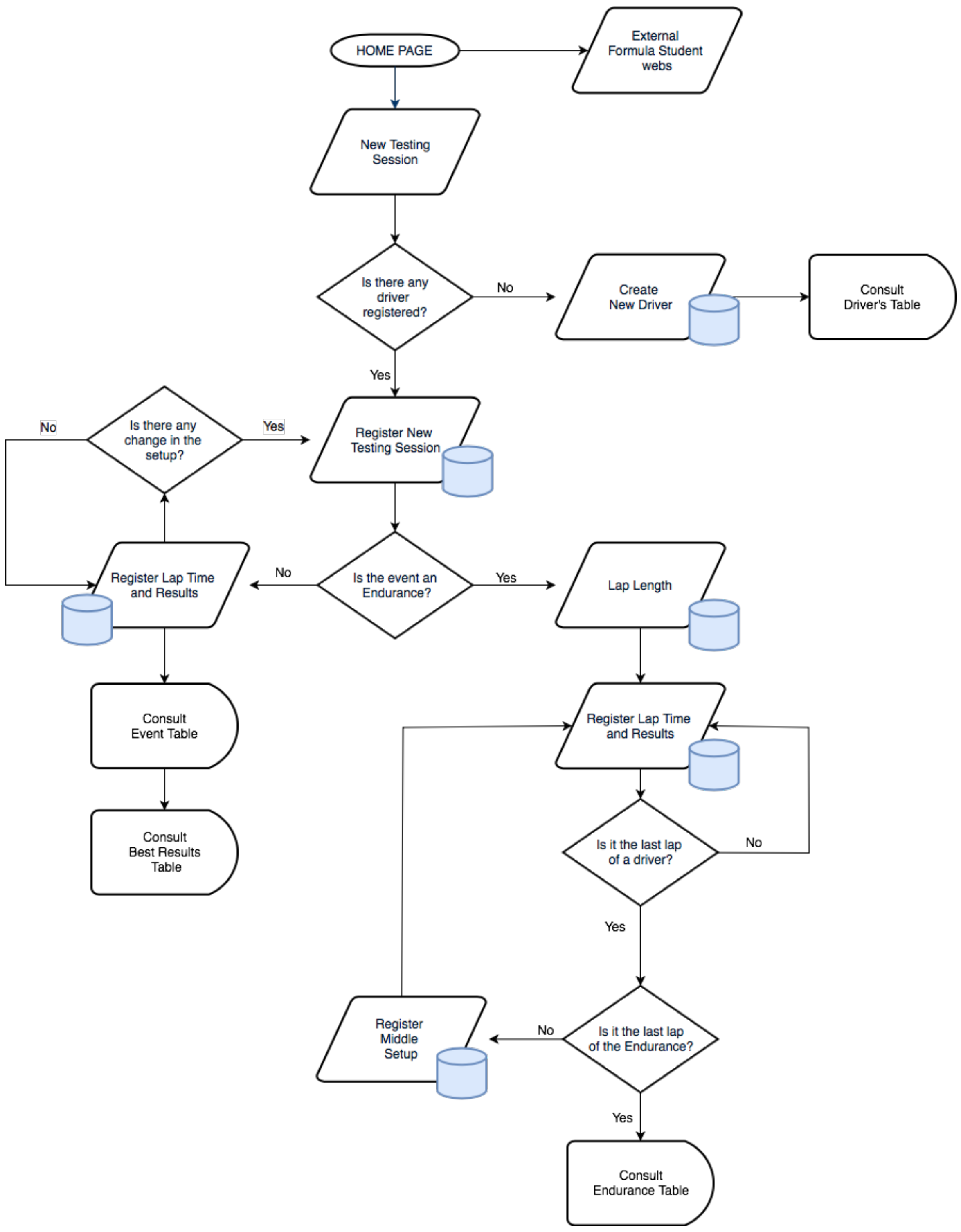

HOME PAGE
External Formula Student webs
New Testing Session
Is there any driver registered?
No
Create New Driver
Consult Driver's Table
Yes
Register New Testing Session
Is there any change in the setup?
No
Yes
Is the event an Endurance?
No
Yes
Register Lap Time and Results
Lap Length
Consult Event Table
Consult Best Results Table
Register Lap Time and Results
Is it the last lap of a driver?
No
Yes
Is it the last lap of the Endurance?
No
Register Middle Setup
Yes
Consult Endurance Table

Figure 5.1 User path along the application following the logical structure starting from the Home page

5.2. Solution internal structure.

To ensure the compliance with the initial requirements, the developed tool has been developed taken advantage of the Django structure. The whole application is divided into different directories:

- EM

Its name stands for ETSEIB Motorsport. This directory contains 4 files required by Django to launch an application. Those files contain information about the database type (in this case, SQLite3), URLs, used templates and some other details.

- Static

Contains information concerning the view of the application. Inside this information is also organized into other directories that contain CSS files, JavaScript files, images and code concerning third party libraries.

- Templates

Contains text files (Django templates) that will be rendered as HTML files and that determine the user interface of the application. The templates are divided itself into two directories: testing, containing the main templates, and blocks, containing files that determine the content of a block that will be integrated in a bigger template view.
The implemented templates and blocks is further explained in section 5.3

- Testing

Contains python files that are the ones handling the actions performed by the user and the response given by the application.

The files integrated in this directory are:

- admin.py
  Allows the access to some tables of the database through the admin interface provided by Django.
- apps.py
  Determines the apps of the project. In this case, the project contains only one app: testing.

- urls.py
  Contains information about the URLs. Each URL is linked to the corresponding view function that will determine the actions to be performed and the content to be displayed.
- views.py
  Contains view functions and classes. The view functions and classes are the ones in charge of handling the requests made by the user. In this project, all views have been implemented as classes (subclasses of *Django.view.generic.TemplateView* class) to provide different responses as function of the request type. Only the home page is implemented with a view function.
  Other helper functions such as the ones in charge of computing statistics are also implemented in this file.
- models.py
  Contains the definition of all Django models used in the project.
- forms.py
  Contains the definition of all Django forms used in the project.
- tables.py
  Contains the definition of all Django tables used in the project.
- resources.py
  Contains the definition of the resources classes. These classes inherit from a superclass of the resources library and handle the generation of .xls files for the corresponding models.
- exports.py
  Contains the functions in charge of generating .csv files for each model.

5.3. User interface.

The user interface has been developed in a responsive way. It means it can be used in devices of different sizes such as laptops, phones, etc. and the interface will adapt to each situation to display the information in a readable way.

The user will be able to perform two main actions: register data or consult it. In both cases the interaction is performed via a simplified interface that shows all the necessary information in a structured way and which is easy to interact with.

The structure of the pages is based on blocks that are shared when possible to reduce the amount of code and simplify the process of performing changes on the view of common elements.

5.3.1. Templates

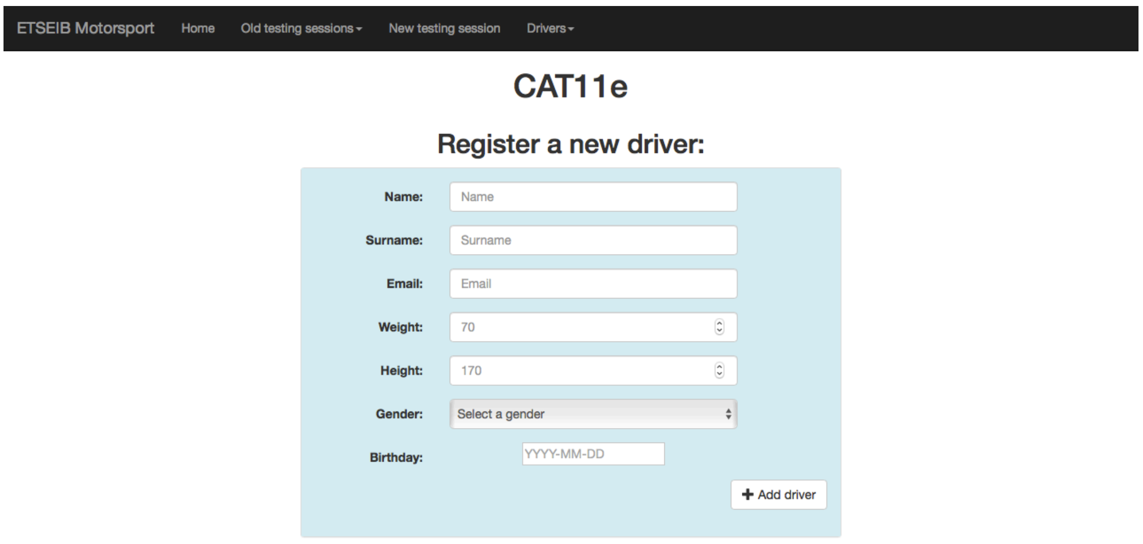
The developed solution counts on the following templates:

- **Home**:

Defines the view of the landing page of the WebApp. It contains the navigation bar block, a welcome message and different boxes with links to related websites of general interest for the team.

- **New Driver**:

View of the page for driver's registration. Contains the view of the form for entering the related data to a driver. The fields of the Driver's model are instantiated using widgets. A Django widget is the representation of an HTML input element. The widget handles the rendering of the HTML, and the extraction of data from a GET/POST dictionary that corresponds to the widget.



CAT11e

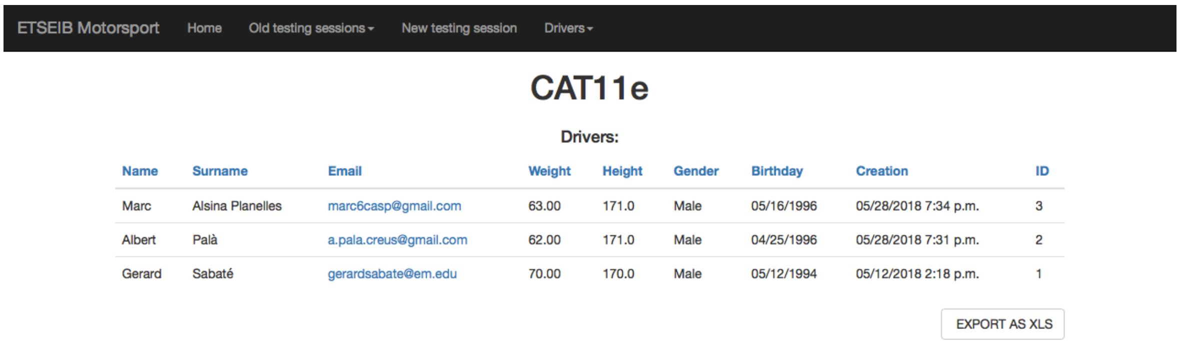
Register a new driver:

Name:	Name
Surname:	Surname
Email:	Email
Weight:	70
Height:	170
Gender:	Select a gender
Birthday:	YYYY-MM-DD

Figure 5.2 View for driver registration. At the top of the page it is displayed the navigation bar. Centered on the page is the form with the different fields requested to create a new driver profile.

- **Drivers**:

View of the table with the existing drivers. The template of the view includes the request of a Django Table. Django takes care of the rendering of the table with the corresponding information what eases the process and simplifies the code.
Just under the table is displayed the button that allows to download the information as an Excel file.

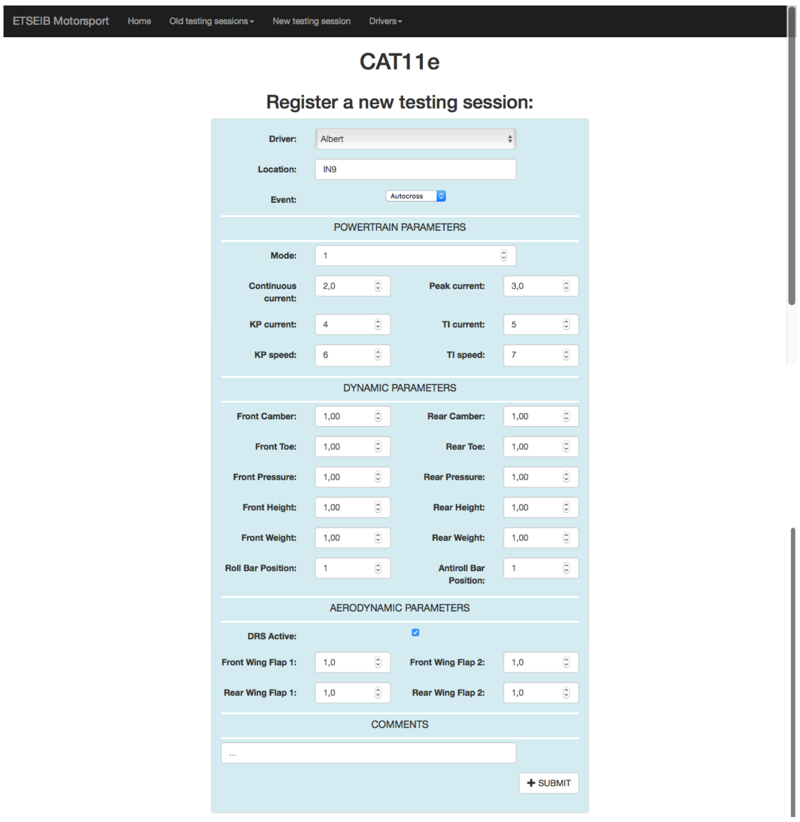


Name	Surname	Email	Weight	Height	Gender	Birthday	Creation	ID
Marc	Alsina Planelles	marc6casp@gmail.com	63.00	171.0	Male	05/16/1996	05/28/2018 7:34 p.m.	3
Albert	Palà	a.pala.creus@gmail.com	62.00	171.0	Male	04/25/1996	05/28/2018 7:31 p.m.	2
Gerard	Sabaté	gerardsabate@em.edu	70.00	170.0	Male	05/12/1994	05/12/2018 2:18 p.m.	1

Figure 5.3 View of the page with the driver's table.

- **New Testing:**

View corresponding to the page where the user creates a new testing session. Its main content is the form where the user can enter all the information for initiating a new testing session. The form is divided into different sections differentiating the type of parameters requested.



CAT11e

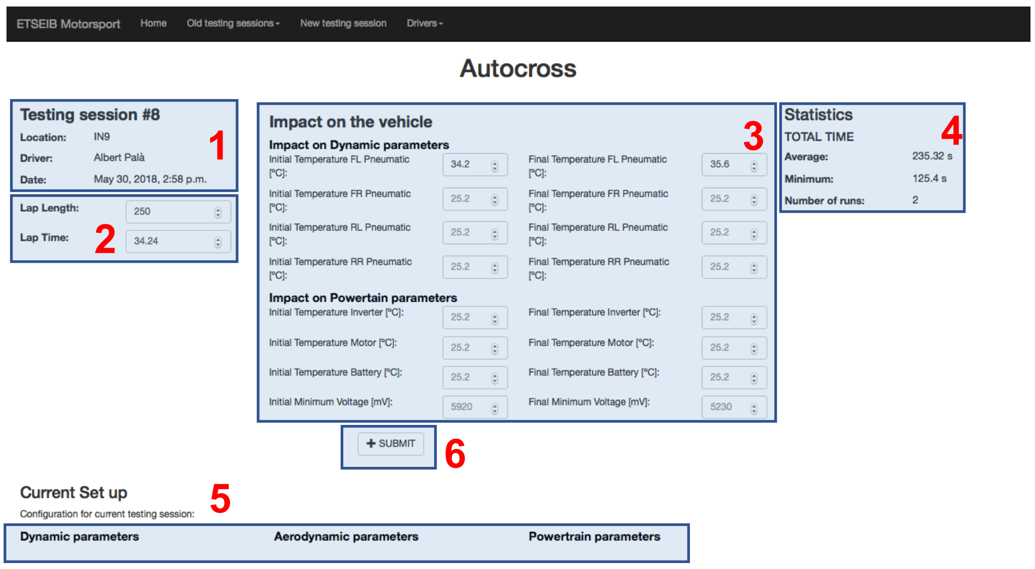
Register a new testing session:

Driver:	Albert		
Location:	IN9		
Event:	Autocross		
POWERTRAIN PARAMETERS			
Mode:	1		
Continuous current:	2,0	Peak current:	3,0
KP current:	4	TI current:	5
KP speed:	6	TI speed:	7
DYNAMIC PARAMETERS			
Front Camber:	1,00	Rear Camber:	1,00
Front Toe:	1,00	Rear Toe:	1,00
Front Pressure:	1,00	Rear Pressure:	1,00
Front Height:	1,00	Rear Height:	1,00
Front Weight:	1,00	Rear Weight:	1,00
Roll Bar Position:	1	Antiroll Bar Position:	1
AERODYNAMIC PARAMETERS			
DRS Active:	✓		
Front Wing Flap 1:	1,0	Front Wing Flap 2:	1,0
Rear Wing Flap 1:	1,0	Rear Wing Flap 2:	1,0
COMMENTS			
...			

Figure 5.4 View of the page concerning the creation of a new testing session

- **Event:**

The template for the view of Acceleration, Skid Pad and Autocross runs. For all cases, different parts of the view can be distinguished (see Figure 5.5 and Figure 5.6). This example will be used to outline the relation between the code and the final view for the user and explain the usage of different Django template features.



Autocross

Testing session #8

Location:	IN9
Driver:	Albert Palà
Date:	May 30, 2018, 2:58 p.m.
Lap Length:	250
Lap Time:	34.24

Impact on the vehicle

Impact on Dynamic parameters

Initial Temperature FL Pneumatic [ºC]:	34.2	Final Temperature FL Pneumatic [ºC]:	35.6
Initial Temperature FR Pneumatic [ºC]:	25.2	Final Temperature FR Pneumatic [ºC]:	25.2
Initial Temperature RL Pneumatic [ºC]:	25.2	Final Temperature RL Pneumatic [ºC]:	25.2
Initial Temperature RR Pneumatic [ºC]:	25.2	Final Temperature RR Pneumatic [ºC]:	25.2

Impact on Powertain parameters

Initial Temperature Inverter [ºC]:	25.2	Final Temperature Inverter [ºC]:	25.2
Initial Temperature Motor [ºC]:	25.2	Final Temperature Motor [ºC]:	25.2
Initial Temperature Battery [ºC]:	25.2	Final Temperature Battery [ºC]:	25.2
Initial Minimum Voltage [mV]:	5920	Final Minimum Voltage [mV]:	5230

Statistics

TOTAL TIME

Average:	235.32 s
Minimum:	125.4 s
Number of runs:	2

Current Set up

Configuration for current testing session:

Dynamic parameters	Aerodynamic parameters	Powertrain parameters

Figure 5.5 View for run registration of an Autocross. The different areas of the page are framed and referenced with a number. The form to register the impact on the vehicle is being displayed while the information about the setup being used is hidden.

The template of the page begins with the head where the different libraries used are stated and the style of some elements is specified. Previously, the type of document (HTML in this case) and the language to be used are specified.

In the head section, it is specified:

- Style
- JavaScript scripts. (such as showing or hiding the tables containing the setup information)
- Libraries (bootstrap)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.
css">
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"><
/script>
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js
"></script>
    <title>Testing session</title>
    <style>
        #info1 {
            font-weight: bold;
            padding-left: 15px;
        }

        #info2 {
            padding-right: 10px;
        }
    </style>
</head>
```

After the head, the body of the page is where the content is specified. The common element for all the pages in the developed web is the navigation bar. It is included as a block, what means that the code in charge of all its specifications is in a different file and is included in the main file through the include function that Django allows to incorporate in the templates.

```
<body>

<-- Navigation bar and event title! -->
<div>
    {% include 'blocks/nav.html' %}
    <h1 align="center">{{ data.event }}</h1>
</div>
```

```
{% load render_table from django_tables2 %}
```

Moreover, the function to render tables is incorporated so that the Django engine knows how to display the content passed as a Django Table to the template.

Next, diverse areas can be distinguished. All of them are integrated inside a form block as their content is changed or evaluated after each user interaction.

1. Block which displays information about the current testing session. In all cases it contains, the testing session's id, location, driver and date.

```
<form method="post">
    {% csrf_token %}
    <div>
        {# Info about the testing session#} – PART 1
        <div class="container col-sm-3" style="padding-left: 20px; font-
size:15px">

            <div class="row">
                <h3 class="col-sm-10" style="font-weight: bold">
                              Testing session #{{ data.id }}</h3>
            </div>

            <div class="row">
                <p class="col-sm-4" id="info1">Location:</p>
                <p id="info2">{{ data.location }}</p>
            </div>

            <div class="row">
                <p class="col-sm-4" id="info1">Driver:</p>
                <p id="info2">{{ data.driver.name }}
                              {{ data.driver.surname }}</p>
            </div>

            <div class="row">
                <p class="col-sm-4" id="info1">Date:</p>
                <p id="info2">{{ data.date }}</p>
            </div>


            {# Request information about the time and lap length
(depending on the event) #} – PART 2
            {% include req %}


        </div>
```

2. Block which registers information about the performance of the car. The information requested is different for each event, reason why the content of this area is passed to the template as a variable. The block is included thanks to the include function.

```
{% include req %}
```

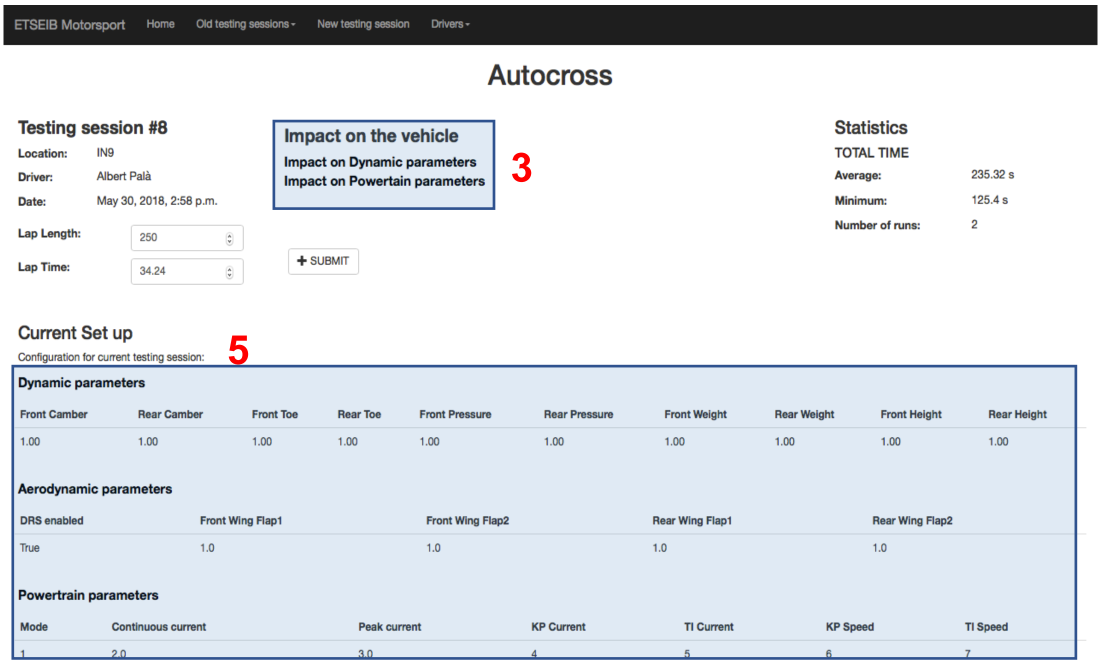
3. Block which registers the information regarding the impact of the run on the different systems of the vehicle. The main area is divided into two different blocks depending on the studied system. Both blocks can be hidden by pressing its corresponding title. This possibility has been incorporated to have a sparser vision, especially when it is being executed from a small device such as a phone.

```
{# Request information about the impact on the vehicle #}
<div id=Results class="container col-sm-6" style="padding: 10px">
    {% include 'blocks/results.html' %}
</div>
```

4. Statistics block. The information displayed is different in the case of Skid Pad as each run is composed of 4 turns and therefore, 4 different times are registered to evaluate the performance of the car.
   Again, as the information is variable for each event, the content to be displayed is passed as a variable to the template.

```
{# Display information about the statistics of the event #}
<div class="container col-sm-3" style="font-size: 15px">
    {% include stat_view %}

</div>
```



Autocross

Testing session #8

Location:	IN9
Driver:	Albert Palà
Date:	May 30, 2018, 2:58 p.m.
Lap Length:	250
Lap Time:	34.24

Impact on the vehicle

Impact on Dynamic parameters
Impact on Powertain parameters

Statistics

TOTAL TIME

Average:	235.32 s
Minimum:	125.4 s
Number of runs:	2

Current Set up

Configuration for current testing session:

Dynamic parameters

Front Camber	Rear Camber	Front Toe	Rear Toe	Front Pressure	Rear Pressure	Front Weight	Rear Weight	Front Height	Rear Height
1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Aerodynamic parameters

DRS enabled	Front Wing Flap1	Front Wing Flap2	Rear Wing Flap1	Rear Wing Flap2
True	1.0	1.0	1.0	1.0

Powertrain parameters

Mode	Continuous current	Peak current	KP Current	TI Current	KP Speed	TI Speed
1	2.0	3.0	4	5	6	7

Figure 5.6 View for run registration of an Autocross. The form to register the impact on the vehicle is hidden while the information about the setup being used is displayed.

5. Block which displays information about the current setup being used. It also incorporates the possibility to hide the different categories of information as it will not be necessary to display it during the whole run but just on special occasions. This block is also a variable of the template.

```
{# Display information about the setup being used for the current
run #}
<div id=Setup class="container col-sm-12" style="padding: 20px">
    {% include 'blocks/setup.html' %}
</div>
```

6. Finally, the submit button allows the user to save the information registered and refresh the statistics. Once the button is pressed, the information provided by the user is registered and the page is cleared, displaying again the fields of the form empties to be filled again.

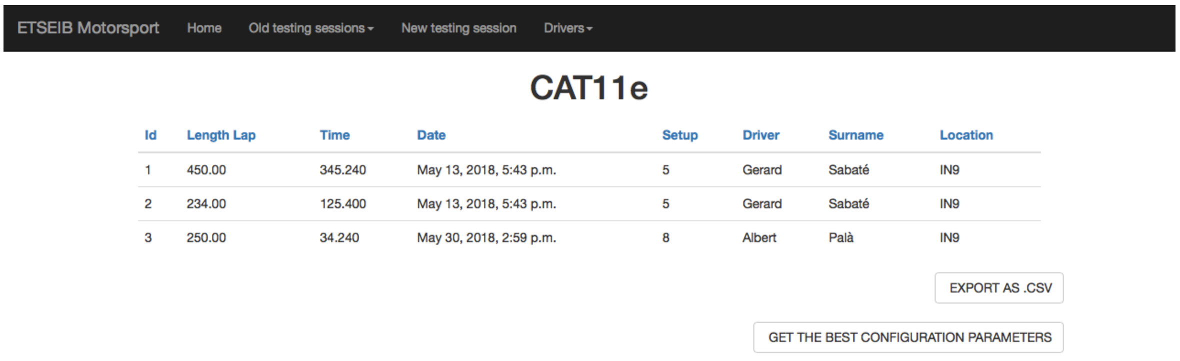
```
      {# Button to submit information #}
      <div class="form-group">
          <div class="col-sm-4">
              <p></p>
          </div>
          <div class="col-sm-8" style="padding-right: 10px">
              <button name="SUBMIT" type="submit" class="btn btn-default
      preview-add-button">
                  <span class="glyphicon glyphicon-plus"></span>
                              SUBMIT </button>
          </div>
      </div>
</form>

</body>
</html>
```

Finally, the tags to close the form, the body and the html file itself are included.

- **Old Testing:**

Template for the view of the information of old testing sessions. It is shared across all events (included Endurance). The information displayed consist of a Django Table with all the instances of the model corresponding to the requested event. At the end of the table, there is a button to download the information as .csv file. Finally, except in the case of the Endurance, there is another button to obtain the setup corresponding to the best result obtained in the consulted event. As this button is displayed based on the event consulted, it is given to the template as a variable which content is an HTML block.



CAT11e

Id	Length Lap	Time	Date	Setup	Driver	Surname	Location
1	450.00	345.240	May 13, 2018, 5:43 p.m.	5	Gerard	Sabaté	IN9
2	234.00	125.400	May 13, 2018, 5:43 p.m.	5	Gerard	Sabaté	IN9
3	250.00	34.240	May 30, 2018, 2:59 p.m.	8	Albert	Palà	IN9

Figure 5.7 View of the results from the different testing sessions from the Autocross event.

- **Best Results:**

The template for the display of the setup corresponding to the best result obtained on a particular event. As in the case of the view for registering runs of an event, the different categories of the setup can be shown or hidden as for the user preferences. This option eases the localization of the desired information, as displaying all the information at the same time could be harder to read, especially with small devices.

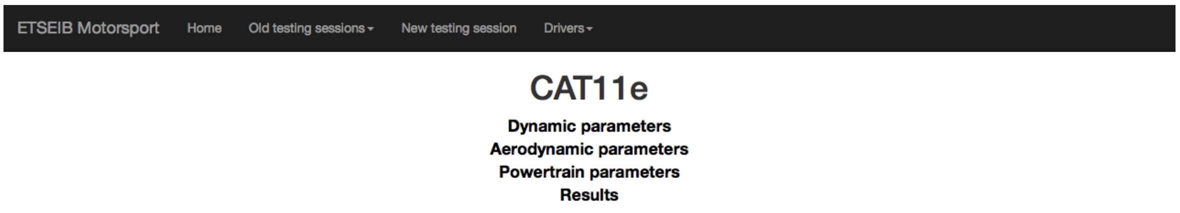
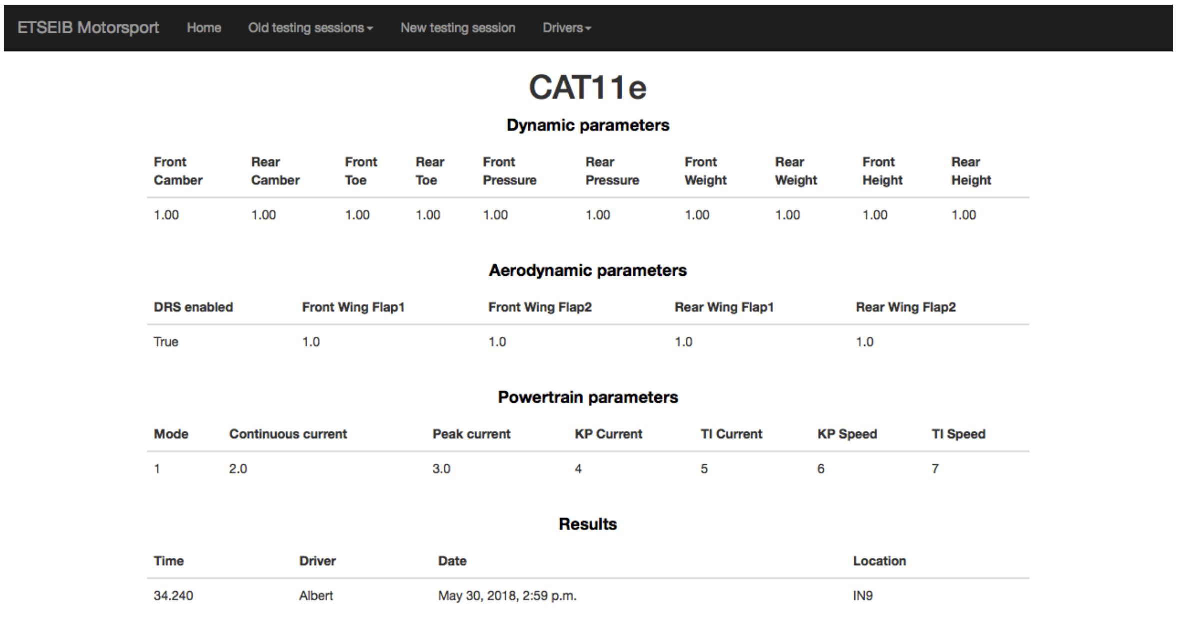


Figure 5.8 View of the Best Results page with all fields hidden



CAT11e

Dynamic parameters

Front Camber	Rear Camber	Front Toe	Rear Toe	Front Pressure	Rear Pressure	Front Weight	Rear Weight	Front Height	Rear Height
1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Aerodynamic parameters

DRS enabled	Front Wing Flap1	Front Wing Flap2	Rear Wing Flap1	Rear Wing Flap2
True	1.0	1.0	1.0	1.0

Powertrain parameters

Mode	Continuous current	Peak current	KP Current	TI Current	KP Speed	TI Speed
1	2.0	3.0	4	5	6	7

Results

Time	Driver	Date	Location
34.240	Albert	May 30, 2018, 2:59 p.m.	IN9

Figure 5.9 View of the Best Results page with all fields displayed

5.3.2. Blocks

Apart from the templates, the interface shares some common blocks along the pages the user can visit. The purpose of using blocks is to simplify some templates and reuse code in order to decrease the possibility of errors and simplify the process of correction when needed.

The current implemented blocks are:

- Navigation Bar: block shared across all the pages. It aids the user to access the different pages of the web.
- Acceleration Request; Skid Pad Request; Autocross Request: these three blocks display the corresponding fields to be filled to register the performance of the car on a run. They can only be used one at a time as they are displayed in the same area of the Event view.
- New Testing Form: contains the structure of the form to register a new testing session. It is included as a separated block, although not being a variable content due to its size and to ease error detections or future changes in the form.
- Setup: block controlling the display of the information related to the current setup using on a run.
- Results: block containing the form referring to the impact of a run on the vehicle' systems.

- Statistics and Statistics Skid Pad: block that contains the information to be displayed in the statistics area. For Acceleration, Autocross and Endurance, “Statistics” is the block used; in the case of Skid Pad the alternative block is the one being called.
- Best Configuration (button): contains the information related to the button Best Configuration that is shared across the different Old Testing views.

5.4. Registration of data

As explained in 3.4.1 one of the main actions to be performed by a user in the registration of testing sessions. This action requires the registration of different types of data. To cope with this, each type of data is associated with a Django Model.

5.4.1. Django Models

A model is the single, definitive source of information about the data. It contains the essential fields and behaviours of the data to be stored. Each model is linked to a table in the database, but instances of the models can be related through some of their fields. Detailed explanation of Django models can be found in section 4.2.2

Each model is presented as a subclass of *Django.Model [5]* class where each attribute represents a database field and is mapped to a database column. Additionally, they can contain methods as normal python classes.

Each attribute of the model is an instance of the appropriate Field class of Django. The field class types allow to determine the kind of data related to the field (text, integer, float, etc.), the way it would be rendered and some minimal validation requirements (e.g. do not accept letters for an integer field).

Some fields of the models relate to other models’ instances. This kind of relations is called One-to-Many relations. The One-to-Many relation is explained later with an example.

The models implemented in the developed app are:

- Driver:
  Contains information related to the driver that can be interesting for the team.

- Testing:
  Contains information about the setup of the car. As different types of parameters conform the final setup of the car, they have been divided into three different categories to ease the handle of possible future changes in the model fields. The three models from which Testing inherits all the parameters are:
    - Dynamic Parameters
    - Aerodynamic Parameters
    - Powertrain Parameters

  These three models cannot be reached individually by the user, but only through the Testing model. Moreover, it contains additional information such as the date, location and the driver performing the testing session (field related to Driver's model instance).

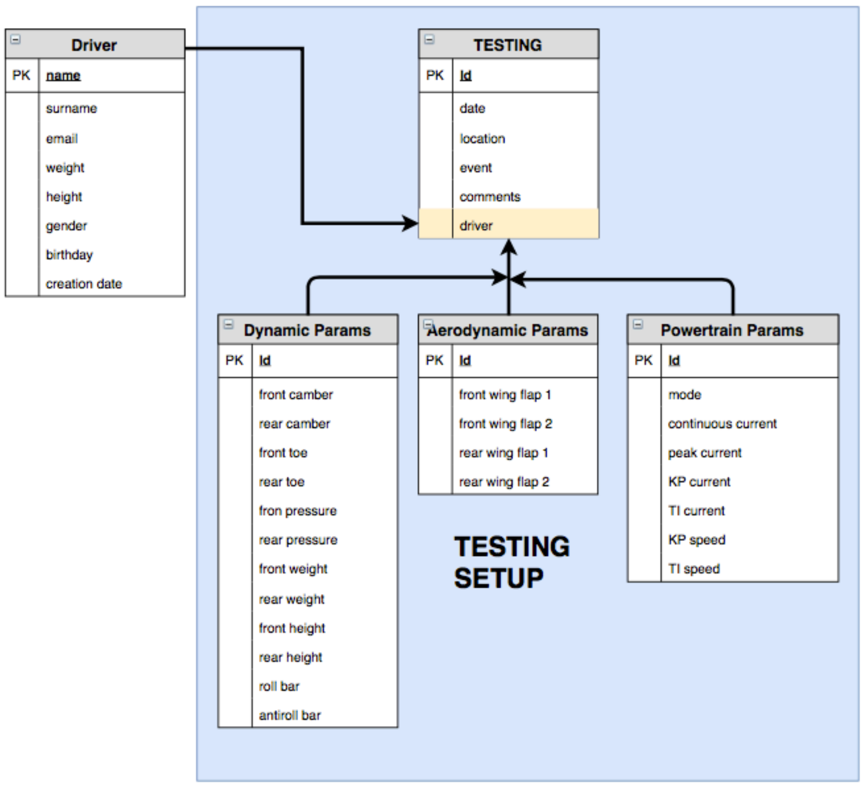


Figure 5.10 Testing and Driver Models. The figure shows the different fields and tables composing each model and the relations between them

- Results:
  Contains information about the impact of a run in some systems of the car. Although including data of different nature, it has not been divided into sub-models as *Testing* as the quantity of parameters was not as big as in the previous case.

- Acceleration:
  Contains information about the time performed in each run. Other information associated is the date of the run, the setup used (related to a Testing's model instance) and the impact on the vehicle (related to a Results' model instance).

- Skid Pad:
  Contains information about the time performed in each run. For each run the time required for each lap (two clockwise and two counters clockwise) is independently stored as well as the final computed time. Other information associated is the date of the run, the setup used (related to a Testing's model instance) and the impact on the vehicle (related to a Results' model instance).

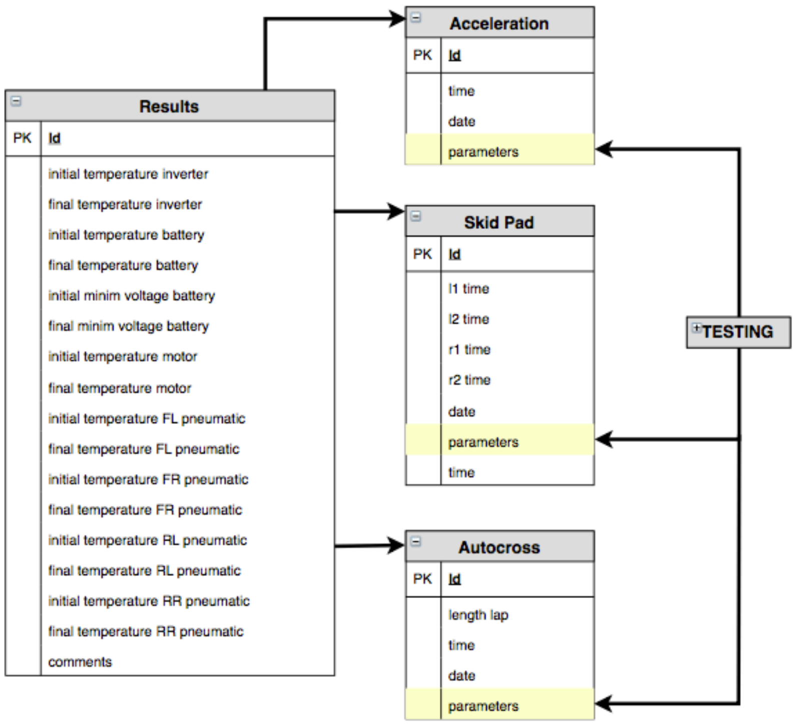

Results
PK Id
initial temperature inverter
final temperature inverter
initial temperature battery
final temperature battery
initial minim voltage battery
final minim voltage battery
initial temperature motor
final temperature motor
initial temperature FL pneumatic
final temperature FL pneumatic
initial temperature FR pneumatic
final temperature FR pneumatic
initial temperature RL pneumatic
final temperature RL pneumatic
initial temperature RR pneumatic
final temperature RR pneumatic
comments
Acceleration
PK Id
time
date
parameters
Skid Pad
PK Id
l1 time
l2 time
r1 time
r2 time
date
parameters
time
Autocross
PK Id
length lap
time
date
parameters
TESTING

Figure 5.11 Results, Acceleration, Autocross and Skid Pad Models. The figure shows the different fields and tables composing each model and the relations between them.

- Autocross:
  Contains information about the time performed in each run and the length of the lap. Other information associated is the date of the run, the setup used (related to a Testing's model instance) and the impact on the vehicle (related to a Results' model instance).

- Endurance:
  Endurance event is a special case as it implies the participation of two drivers, possible changes in the setup of the vehicle in the middle of the event and a fixed number of laps depending on the lap length.
  To ensure the correct registration of the data according with the event schedule, it is registered following the next procedure:

  1. Registration of an Endurance event creating an instance with the initial setup of the vehicle and the first driver who will take part in the event.
  2. Registration of lap length. This allows to compute the number of laps to be performed by each driver.
  3. Registration of lap time and impact as instances of Lap Model.
  4. Registration of changes in the setup and new driver. This step is forced by the system whenever the number of laps to be performed by the first driver is reached.
  5. Registration of lap time and impact for the second part of the event.

  Once the number of required laps is reached by the second driver, the Endurance event is closed and no more data concerning it can be registered, it can only be consulted.

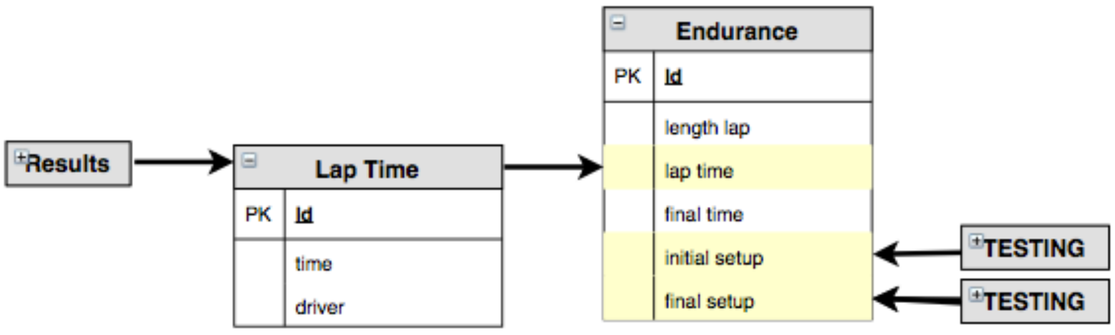


Figure 5.12 Lap Time and Endurance Models. The figure shws the different fields and tables composing each model and the relations between them

Whenever an instance of a model is created, a new row in the related table of the database is created. In that way, data is stored sequentially and can be easily consulted when the user requires.

Example code of a Model class:

```
class Skid_Pad(Results):
    id = models.AutoField(primary_key=True)
    l1_time = models.DecimalField(decimal_places=3, max_digits=5,
default='')
    l2_time = models.DecimalField(decimal_places=3, max_digits=5,
default='')
    r1_time = models.DecimalField(decimal_places=3, max_digits=5,
default='')
    r2_time = models.DecimalField(decimal_places=3, max_digits=5,
default='')
    date = models.DateTimeField(auto_now=True)
    params = models.ForeignKey(Testing, on_delete=models.CASCADE,
null=True, blank=True)

    def __str__(self):
        return str(self.id)

    @property
    def time(self):
        return float("{0:.3f}".format(float(self.l2_time) +
float(self.r2_time)))
```

From the *Skid_Pad* model we can analyse the structure of a model class in the developed app.

First, we can see *Skid_Pad* is a subclass of the class *Results* which contains many fields of the type "*models.DecimalField()". As a consequence, it contains all the fields specified in the parent class.*

From the fields not inherited from Results, we can distinguish different types:

- 'id' field. It is a *models.AutoField()* generated automatically for every instance. It also represents the access key to the instance.
- 'xx_time' field. We have 4 *models.DecimalField()* that register the time per each lap (right or left, first or second) in the skid pad event. These fields are defined to expect as input a 5-digit number with a maximum of 3 decimals.
- 'date' field. It is a *models.DateTimeField()* generated automatically when an instance is created.

- ‘params’ field. It is a *models.ForeignKey()* field. It means it expect as an input an instance of another model, in that case, an instance of the Testing model. This relation is called One-to-Many relation. It acts in the following way: the field ‘params’ can only be related to one instance of the Testing model. On the other hand, each instance of the Testing model can be requested by many other models and instances of them.

Apart from the fields, a model can also contain properties. In this case, *‘time’* is defined as a property and it represents the final computed time for the Skid Pad event.

5.5. Consultation of data.

5.5.1. Onsite. Data Tables.

Each model of the previous ones is linked to a Django Table [6]. Each table is created from a model and populated with the existing instances of it whenever it is required. Nevertheless, not all the models have an associated table model, but only the ones that can be consulted by the user.

In the case of the developed app, the following table models have been created:

- Driver Table
- Testing Table
- Acceleration Table
- Skid Pad Table
- Autocross Table
- Endurance Table

Some of the table models, especially the ones concerning data from events, have fields related to other models as a *OneToMany* fields. This information is not initially incorporated in the table as in not specified as a field of the main model. To cope with this limitation, some extra columns have been added to the table model using the class Column of the Django Table library.

Example code of a Table class:

```
class SkidPadTable(tables.Table):
    # add custom columns to table (concerning Foreignkey parameters)
    setup = Column(accessor='params.id', verbose_name='Setup')
    driver = Column(accessor='params.driver')
    driver_ = Column(accessor='params.driver.surname')
    date = Column(accessor='date')
    place = Column(accessor='params.location')
    time = Column(accessor='time', verbose_name='Total time')
    class Meta:
        #define the Model related to the table
        model = Skid_Pad
        # template associated for the display of the table
        template_name = 'django_tables2/bootstrap-responsive.html'
        # define fields to be displayed
        fields = ('id', 'l1_time', 'l2_time', 'r1_time', 'r2_time',
                  'time', 'date', 'setup', 'driver', 'driver_', 'place')
```

The fields '*id*' and '*xx_time*' are actual fields of the Skid Pad. The table fields defined as *Columns()* are special cases. In that case, they need to be added as they require deeper information of a related model instance (e.g. driver_ is a field that contains the surname of the driver. The surname of the driver is registered in a Driver model instance and related to the Skid Pad model through a Testing model Instance that contains a Driver instance among its parameters.)

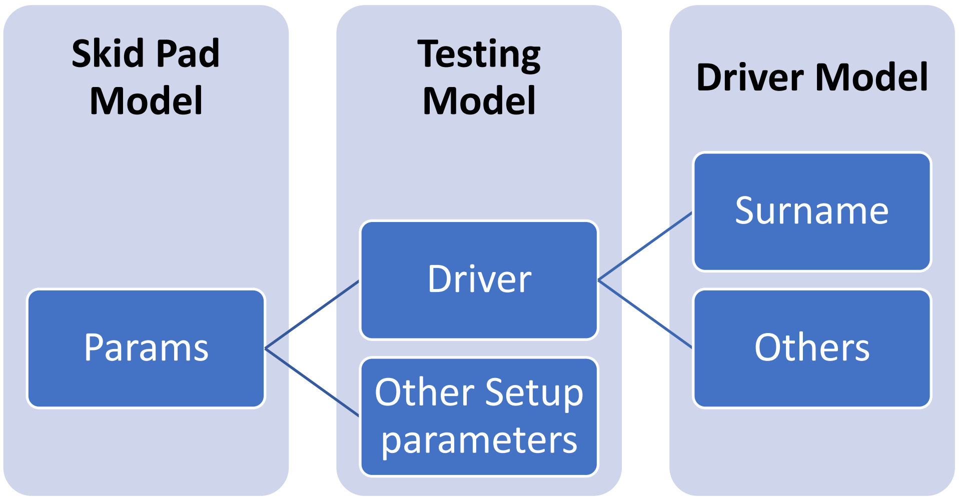


Figure 5.13 Relation between different fields along different models. Params is a Field of the model Skid Pad that contains an instance of the model Testing. One of the fields of the Testing model is Driver that contains an instance of the model Driver. Finally, the Driver model contains the Field called Surname that was instantiated in Skid Pad Table as Column.

5.5.2. Downloading data.

In many cases, the data recorded needs to be downloaded to perform further analysis together with data recorded from the vehicle's sensors.

The data available to be downloaded concern the following models:

- Driver
- Testing
- Acceleration
- Skid Pad
- Autocross
- Endurance

The data is downloaded as .csv file. In each case, each row of the file concerns an instance of the model while each column represents a field of the model or related models.

For the generation of .csv fields, two approaches have been taken:

5.5.2.1. Import_Export Library Approach.

The *import_export* library [7] provides an easy way to generate a .csv file from a Django Model. Its main advantage is that it is not necessary to specify the columns or fields to be included in the file what reduces the need of code to generate a .csv from a model. On the contrary, it does not provide the flexibility to include extra fields what can become a great disadvantage.

The models to be exported using this approach are:

- Drivers
- Testing

To generate a .csv file associated to a model, it is necessary to create a class who inherits from ModelResources class from the import_export library.

An example is stated below:

```
class DriverResource(resources.ModelResource):
    class Meta:
        model = Driver
```

As it can be seen, it is really simple as it only requires specifying the associated model. Later, to generate the .csv file when the user request for it, the code is specified below:

```
def export(self):
    #specify the Resources model associated
    person_resource = DriverResource()
    #generate dataset instance
    dataset = person_resource.export()
    #generate the csv file
    response = HttpResponse(dataset.csv,
content_type='application/vnd.ms-excel')
```

```
    #populate the csv file
    response['Content-Disposition'] = 'attachment;
filename="drivers.csv"'
    return response
```

The great advantage is the easy implementation when the information required can be all found as fields of the related Django Model.

5.5.2.2. Django Utils Approach.

Despite the great advantages of the earlier approach, its rigidness does not allow to add custom fields as columns in the dataset, what can result in a loss of information for the user.

To cope with this limitation, another approach is implemented that requires specifying all the desired fields and therefore, larger amount of code, but yield self-designed structure files.

The function implemented to generate a .csv file containing the Acceleration instances is presented below:

```
def export_CSV_acc(queryset, event):

    response = HttpResponse(content_type='text/csv')

    response['Content-Disposition'] = 'attachment;
filename={}.csv'.format(event)

    writer = csv.writer(response, csv.excel)

    response.write(u'\ufeff'.encode('utf8'))

    # define first row -- Headers of columns

    writer.writerow([
        smart_str(u"ID"),
        smart_str(u"Time"),
        smart_str(u"Lap Length"),
        smart_str(u"Date"),
        smart_str(u"Driver"),
        smart_str(u"Location"),
        smart_str(u"Params"),

… all the missing fields are specified in the original code that can be
found in the annex of this report. …
    ])
    # write each acceleration/autocross run in a new row following the
column order set before

    for obj in queryset:

        writer.writerow([

            smart_str(obj.id),
            smart_str(obj.time),
            smart_str(obj.length_lap),
            smart_str(obj.date),
            smart_str(obj.params.driver),
            smart_str(obj.params.location),
            smart_str(obj.params),

               … all the missing fields are specified in the original
code that can be found in the annex of this report. …

        ])
    return response
```

5.6. Data manipulation.

As previously mentioned, deeper analysis of the recorded data is not performed within the application frame, but later with other software tools designed for that aim. Nevertheless, a few analyses can be made to extract quick conclusions about the performance of the car.

5.6.1. Lap time analysis.

An interesting analysis to be performed after each run is the comparison of the result with the best result obtained up to the moment. This allows to determine whether it has been a good run or not (this allows to take further conclusions to the team).

With the purpose to provide a reference data to the team when performing the runs, a function that provides some statistics has been implemented.

Statistics provides the number of runs performed up to the moment, the minimum time and the average time for the run.

In the case of the Skid Pad event, it is slightly modified to cope with the structure of different laps per run and provide further information, about the full performance on the run but also about the performance in each turning sense.

The code of the ‘statistics’ function is shown below:

```
def statistics(obj):
    if not obj:
        avg = '-'
        min = '-'
        runs = 0
        return list((avg, min, runs))

    min = float(obj.aggregate(Min('time'))['time__min'])
    avg = float(obj.aggregate(Avg('time'))['time__avg'])

    min = float("{0:.3f}".format(min))
    avg = float("{0:.3f}".format(avg))
    runs = obj.count()
    return list((avg, min, runs))
```

5.6.2. Parameters of best result.

This functionality provides easy access to the setup of the car related with its best result in a concrete event up to the moment. This is useful for the team if they want to reproduce the tests performed with better results as it does not require consulting big amounts of data to look for the best performance, but it can be obtained by just a click. The best performance is known as the run with the lowest time.

For the Endurance event, not only the time required to complete the event matters, but other factors may be more important to consider too. Therefore, the “Best Parameters” option is not available as the selection of a proper configuration requires further analysis from the team.

The code of the function that provides the configuration for the best results can be found below:

```
def best_results(request, event):

    if event == "skidpad":
        stats = statistics_sk(Skid_Pad.objects.all())
        runs = stats[2]

        if runs == 0:
            # Redirects to New Testing Session as there are no results
about it.
            return redirect('../{}'.format(event))

        else:
            min_l = stats[4]
            min_r = stats[6]
            min_time1 = Skid_Pad.objects.filter(l2_time=min_l)[0]
            min_time2 = Skid_Pad.objects.filter(r2_time=min_r)[0]
            if min_time1.time < min_time2.time:
                data = min_time1

            else:
                data = min_time2

            return render(request, 'testing/best_results.html', {'data':
data, })

    else:
        if event == "acceleration":
            objs = Acceleration.objects

        elif event == "autocross":
            objs = AutoX.objects

        else:
            objs = Lap_time.objects
        stats = statistics(objs.all())

        if stats[2] == 0:
            # Redirects to New Testing Session as there are no results
about it.
            return redirect('../{}'.format(event))
        else:
            min_time = stats[1]
            data = objs.filter(time=min_time)[0]

            return render(request, 'testing/best_results.html', {'data':
data, })
```

6. Test and validation.

The tool developed as a result of this project will be tested by the Formula Student team ETSEIB Motorsport, during the months of July and August. Before its test on track, different validation tests have been performed in order to detect possible failures of the system and develop a friendlier environment for the user.

Some of the tested features are:

- Correct saving of all data introduced. Once the user presses the “Submit” button, if the data is valid, it is always correctly saved, creating an instance of the corresponding model. This has been checked every time a new model was created.
- Introducing wrong data type. In case the user introduces the data in a correct format or with a value out of the predefined range, the form raises an error thanks to the data validation function that Django incorporates. In this case, it has been decided to redirect the user to the same form indicating that some of the values introduced were not correct.
- Correct pre-population of forms when an event is accessed directly from its URL instead of by creating a new event instance. That is interesting when on the same testing session/day, different events are carried on. For example: morning session – acceleration; afternoon session – autocross. After performing some autocrosses, the team decides to go back to perform accelerations at the end of the afternoon to test the same configuration used during the morning session but with the battery almost discharged. In this case, it is interesting to access directly to the last acceleration configuration instead of creating a new testing session with the same parameters. For that, the user can access through the URL corresponding to the acceleration event and the setup for the next runs will be pre-populated with the one used in the last acceleration run, it means, the one used at the end of the morning session.
- Correct generation of output files. The files generated (.xls or .csv) are available to read with third party software and they contain the correct data with all specified fields and all the corresponding entries of the model.

6.1. Detected errors and correction

Some of the errors detected during the validation test were:

- Usage of the last overall setup registered for an event when it is accessed through its URL. This error has been corrected to set as corresponding setup the last one used for that event instead of the last overall.
- Accessing to the “New testing session” without preregistered drivers. In this case, it will be impossible to create a new testing session as there will be no drivers to be assigned to it. It has been corrected so that the user is redirected to the “Register new driver” page.
- Accessing an event through its URL without any preregistered setup for it. In this case, there will not be an available setup to assign to the corresponding runs of the event, therefore, the user is redirected to the “New testing session” to first create a session for the corresponding event.
- Accessing the “Best results” page for an event when no runs have been performed. As there is no available data to compare, the user will be redirected to the “New testing session” page to create a new session for this event.

7. Economic study.

In this section, the costs related to the development of the project will be exposed. The main costs taken into account are the human resources (hours invested in the project) and both software and hardware resources.

7.1. Human Resources

The hourly cost of an engineer working on the project can vary depending of the performed tasks. Based on the data extracted from the INE (National Institute of Statistics) [10] and from a jobs internet site [11], the average annual salary of a web developer in Spain is about 60.000€, varying on the experience, type of contract, sex and size of the company, among other factors.

For the analysis and documentation tasks, the hourly rate will be considered the same as the tasks have been performed by the same person and it can be considered as an autonomous worker developing the whole project.

For a salary of 60.000€/year, the corresponding hourly rate is of 30€/h.

Finally, concerning the amount of time dedicated to each task, further details can be reach at section 9.

Concept	**Unitary Price [€/h]**	**Quantity**	**Amortization**	**Final Price [€]**
Analysis	30,00	82	NA	2.460,00
Tool development	30,00	198	NA	5.940,00
Documentation of the final project	30,00	50	NA	1.500,00
TOTAL		330	NA	9.990,00

Table 7.1 Detailed Human Resources cost of the project

7.2. Hardware and Software

The cost of the hardware and software tools employed for the development of the project are detailed in the following table:

Concept	**Unitary Price [€]**	**Quantity**	**Amortization [%]**	**Final Price [€]**
Laptop (MacBook Pro Mid 2012)	1450,00	1	8	120,83
Pycharm Licence	130,00	1	50	65,00
Microsoft Office Licence	149,00	1	8	12,42
Python and Libraries Licences	0	1	NA	0,00
TOTAL	-	-	-	198,25

Table 7.2 Detailed Hardware and Software costs

7.3. Final Cost

Other costs such as electricity, internet connection of working space have not been considered as the project has been developed in a public space (EPF Lausanne library) that provides all these factors without cost and that can be used by anybody without the need of being a former student from the university.

Finally, the total cost of the project is 10.098,25€

8. Environmental impact.

The environmental impact of this project comes essentially as a result of the energy consumption of the tools used for its development.

The impact calculations are shown in the following table:

Concept	**Power Consumption**	**Time**	**Energy Consumption**	**CO_2 emission**
Analysis, development and documentation	270 W	330h	89,1kWh	34,93kg

Table 8.1 CO_2 emissions calculation. Data retrieved from: Power consumption of the laptop: Apple [12]; Emission factor: Generalitat de Catalunya [13]

The impact generated in terms of CO_2 emissions is equivalent to driving 310km with a vehicle of medium size (data extracted from the IDEA [14]).

The impact caused by energy consumption of light and internet is not integrated in the final amount of CO_2 emissions as the project itself has been developed without generating an extra demand of those two. It means, the energy used to illuminate the space or provide it with internet connection will have remained the same with or without the development of this project.

9. Planification.

This project has taken place between February 2018 and June 2018. During this time, different phases have been carried out: from the study of the problem to the implementation of the final solution, the time has been divided between the different tasks taking into account the complexity of each one and its impact on the final result.

First, and analysis of the problem and study of possible solutions has been carried out. Once the solution of developing a WebApp was chosen, it was necessary a learning period of the required tools to implement this solution.

For the implementation, the tasks have been divided in groups taking into account its similarity and the relation between the different fields. Registration of drivers has been the first feature implemented due to its relationship with all the other features -it is necessary to have drivers already registered to be able to register a testing session-. Next task has been implementing the registration of the setup and therefore, the creation of testing sessions. Once finished, the registration of Acceleration, Skid Pad and Autocross has been handled simultaneously due to the similarity of the structure of the data required by each of those events. Endurance has been the last event to be implemented due to its complexity and requirements in terms of model relations.

After the implementation of the data collection tools, the implementation of the data output has been addressed. Finally, once the project was operative and ready to be used, it has been deployed on Heroku.

After the deployment, a phase of testing and correction of errors has followed. This phase has concluded with the testing of the WebApp in real conditions by the team ETSEIB Motorsport during a testing session.

The writing of the final report has been carried out at the same time of the development of the solution, making special emphasis on periods when one of the previously mentioned phases was completed.

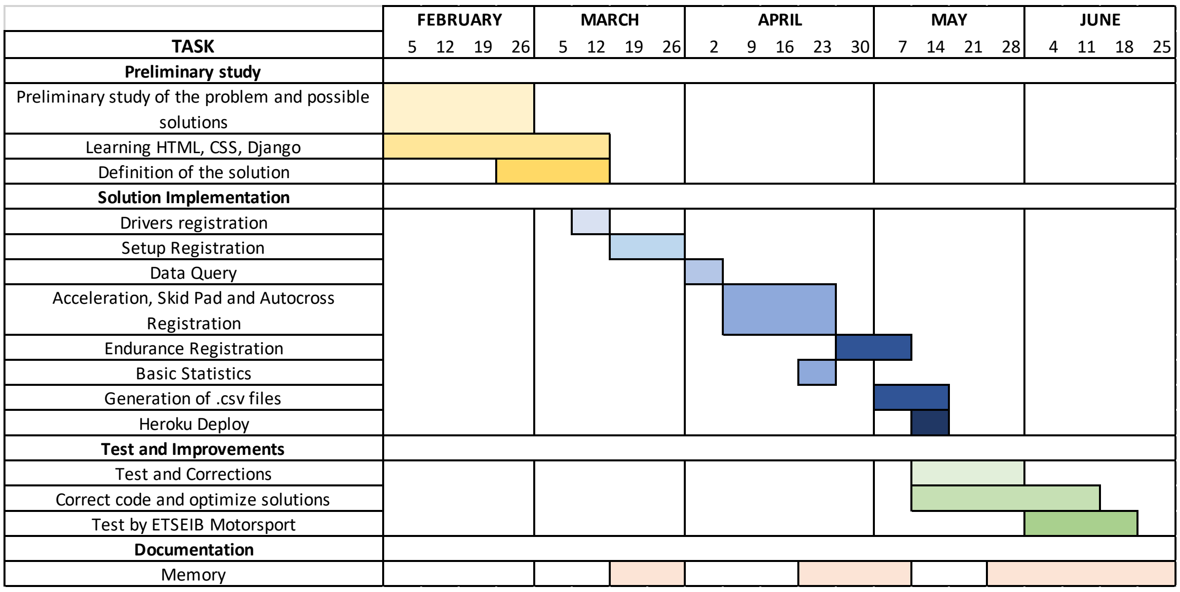


Figure 9.1 Temporal planning of the projec

Conclusions

The main objectives of this project were to develop a tool for collecting data during the testing sessions of a Formula Student prototype. This tool should provide a standardize procedure for the collection of the data and guarantee the access to it along the time. Finally, the requirements set for the tool itself were: the possibility to save data, independently of the device and location, in a common database and the availability to consult it and export it.

After the conclusion of the project, it can be stated that all objectives have been achieved. After an initial analysis of the requirements and the previous procedure, it has been decided to develop a solution in the form of web application. The developed application guarantees a standardized procedure to ensure the correct registration of data for further analysis and comparisons along time. Moreover, it can be accessed from different devices and locations to avoid depending on one device which contains all the data. In this way, every team member is able to perform the registration of testing information with its own mobile or laptop. Finally, the data is available to be downloaded for further analysis with third party software.

The development of this project has also had an educational approach. It has allowed the author to learn new programming languages as well as work with new frameworks such as Django, which is widely used in the industry nowadays. This all has meant the firsts steps of the author in the development of web applications.

Finally, the project has been developed in constant contact with ETSEIB Motorsport as it being the final user of the tool. This has led to proceed in a flexible way, adapting to their requirements and evolving from a simpler solution to a more complete and complex one.

Regarding the points to be improved or continue developing, some analysis tools could be implemented in the same application to integrate the analysis of the registered data and generate informs of each session directly. Also, improvements to the interface or display of the content could lead to a more modern and attractive interface. Finally, after sharing the results of this project with other teams and them showing their interest in this tool, it will be interesting to implement a user manager so that the application could be used by different teams and each one had its own user profile with its data.

Acknowledgements

First, I would like to thank the board of the *Escola Tècnica Superior d'Enginyeria Industrial de Barcelona* for the support given to the Formula Student team ETSEIB Motorsport. Year after year, this project allows a great number of students to get involved in an amazing experience enrich both professional and personally. Especial thanks to Lluís Solano, for its support showed to the team and for its support during the development of this project acting as director.

This project would not have been possible either without all the students that have taken part in the Formula Student team, especially the ones taking part in the development of the CAT10e which set the origins of the project.

Thanks to Pau Argelaguet for the given advice during the project and for always being up to solve my doubts.

Finally, thanks to my family for all the support given during my studies.

ETSEIB

Bibliography

Bibliographic references

[1] FORMULA STUDENT GERMANY (March 10th, 2018) *Disciplines.* Retrieved from: [https://www.formulastudent.de/about/disciplines/]

[2] WIKIPEDIA (April 8th, 2018) *HTML5*. Retrieved from: [https://en.wikipedia.org/wiki/HTML5]

[3] CODEMENTOR (April 8th, 2018) *Python Framework comparison: Django vs. Pyramid*. Retrieved from: [https://www.codementor.io/sheena/django-vs-pyramid-python-framework-comparison-du107yb1c]

[4] BOOTSTRAP (March 22th, 2018) *About Bootstrap*. Retrieved from: [https://getbootstrap.com]

[5] DJANGO MODELS (May 5th, 2018) *Documentation. Django Models.* Retrieved from: [https://docs.djangoproject.com/en/2.0/topics/db/models/]

[6] DJANGO TABLES (May 5th, 2018) *Documentation. Django Tables*. Retrieved from: [https://django-tables2.readthedocs.io/en/latest/]

[7] DJANGO IMPORT-EXPORT (May 5th, 2018) *Documentation, Django Import-Export*. Retrived from: [http://django-import-export.readthedocs.io/en/latest/]

[8] Sun Xu, Wan Li. Espais ETSEIB. Desenvolupament d'una pàgina web amb un framework de Python. Barcelona s.n., 2015

[9] DEVOPEDIA. 2017. (May 19th, 2018) *"Django." Version 17, November 1.* Retrieved from: [https://devopedia.org/Django]

[10] INE 2018 (June 5th, 2018) "*Wages and labour costs".* Retrieved from: [http://www.ine.es/dyngs/INEbase/es/categoria.htm?c=Estadistica_P&cid=1254735976596]

[11] INDEED (June 5th, 2018) Web developer salaries. Retrieved from: [https://www.indeed.es/salaries/Programador-Salaries]

[12] Apple Support (June 5th, 2018) Information about power consumption of different devices.
Retrieved from: [https://support.apple.com/es-es/HT201796]

[13] GENERALITAT DE CATALUNYA (June 6th, 2018) Emission factor related to electrical energy: he electrical mix.
Retrieved from: [http://canviclimatic.gencat.cat/en/redueix_emissions/com-calcular-emissions-de-geh/factors_demissio_associats_a_lenergia/index.html]

[14] IDEA (Instituto para la Diversificación y Ahorro de la Energía) (June 7th, 2018) SEAT vehicles CO_2 emissions. Retrieved from: [http://coches.idae.es]

Complementary bibliography

DJANGO (February, 2018). Documentation referent to the Django framework. Retrieved from: [https://www.djangoproject.com]

FLASK (February, 2018). Documentation referent to the Flask framework. Retrieved from: [http://flask.pocoo.org]

PYRAMID (February, 2018). Documentation referent to the Pyramid framework. Retrieved from: [https://trypyramid.com]

HTML, CSS, JS and BOOTSTRAP tutorials. (February – June, 2018) Retrieved from: [https://www.w3schools.com]