Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Telemática

Trabajo Fin de Grado

Visualizador de datos recogidos por un vehículo autónomo

Autor: David García Ruiz

Tutor: Iván García Daza

Cotutor: Carlota Salinas Maldonado

2021

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Telemática

Trabajo Fin de Grado

Visualizador de datos recogidos por un vehículo autónomo

Autor: David García Ruiz

Directores: Iván García Daza

Tribunal:

Presidente: Ignacio Parra Alonso

Vocal 1º: Javier Alonso Ruiz

Vocal 2º: Iván García Daza

Calificación: .....

Fecha: .....

A nuestros alumnos pasados, presentes y futuros...

"Empieza haciendo lo necesario, luego haz lo posible y de pronto empezarás a hacer lo imposible."
Francisco de Asís

Las base de todo conocimiento.....

"Lo importante no es saber, sino tener el teléfono del que sabe."
Les Luthiers

Agradecimientos

A todos los que la presente vieren y entendieren.

Inicio de las Leyes Orgánicas. Juan Carlos I

Este trabajo es el fruto de muchas horas de trabajo, búsquedas de información y autoformación, así como un afán incondicional de búsqueda de conocimiento.

Quisiera hacer una mención especial Juan Ignacio García García e Ignacio García Ruiz por forzarme a querer seguir aprendiendo en el mundo de la tecnología.

Durante este desarrollo Alejandra Ramos Romero jamás me dejo rendirme y abandonar este proyecto con el fin de culminar este hito en mi carrera profesional.

Finalmente, hay incontables contribuyentes gracias a sus conocimientos que fueron consultados mediante el todo poderoso Google. He intentado referenciar los más importantes en los fuentes de este documento, aunque seguro que he omitido alguno.

Me gustaría añadir que sin ninguno de ellos habría podido realizar este proyecto.

Resumen

Este documento ha sido generado como memoria del trabajo de fin de grado, en el cual se plasma la manera de visualizar los datos recogidos por los diferentes subsistemas de un vehículo autónomo con el fin de representar en tiempo real el estado actual del vehículo, incluso ser capaz de visualizar datos de un trayecto, reproduciendo el viaje transcurrido o viendo la evolución de manera estadística de valores recogidos por el sistema.

El origen de este proyecto aparece de la necesidad de una manera de visualizar los datos que recogía un vehículo autónomo de manera poco optima, se recogía en ficheros locales dentro del propio vehículo.

El modelo de infraestructura completo sobre el que está estructurado el proyecto sigue la modelo básico de cualquier sistema con IoTs:

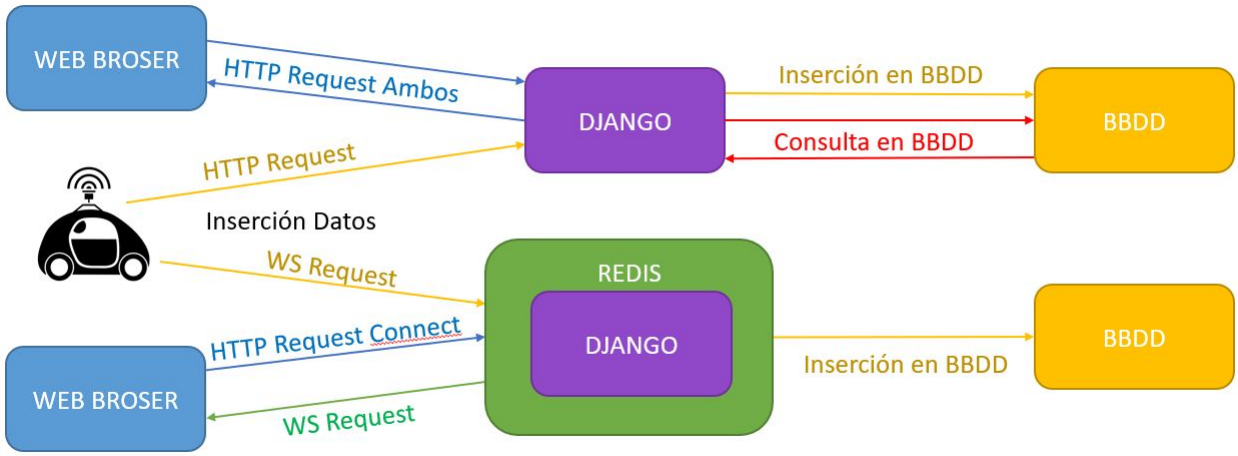

Figura 2: Esquema 1: Modelo de infraestructura para toma de datos de vehículo autónomo.

Para la implementación de la solución será necesario una plataforma basada en una base de datos (MySql) y un servidor (en el que puede estar la misma base de datos) para hacer funcionar la aplicación con el Framework de Django.

Índice general

Índice de figuras

Índice de tablas

Índice de listados de código fuente

Capítulo 1

Introducción

En este proyecto se pretende el diseño de un visualizador de los datos recogidos por los diferente sensores existentes en un vehículo autónomo. Se ha decido utilizar Django[1], para la generación de este interfaz visual.

El proyecto debe contener un visor en tiempo real[2], un sistema que sea capaz de visualizar un histórico con los datos de los viajes realizado y la evolución del viaje mediante gráficas.

Figura 1.1: Imagen de vehículos conectados.

1.1 Estado actual del sistema

Actualmente el vehículo cuenta con un sistema ROS[3] con el cual recoge los datos de los diferentes sensores del vehículo y los guarda en ficheros de texto con formato JSON[4].

[1] Django es un Framework de desarrollo web de código abierto, con lenguaje de programación basado en Python, que respeta el patrón de diseño conocido como MVC (Modelo-Vista-Controlador).

[2] Tiene que poder visualizarse con la menor demora posible entre el vehículo y la página que represente los datos.

[3] ROS (Robot Operating System) es un Framework para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo

[4] JSON (JavaScript Object Notation) es un formato de intercambio de datos con un aspecto de árbol con referencia de clave-valor para cada uno de los parámetros.

1.2 Objetivos

El objetivo de esta implantación es poder visualizar datos en tiempo real, poder repetir la visualización de viajes anteriores y sacar valores mediante gráficas estadísticas de la evolución del vehículo autónomo durante un trayecto. Esto facilitaría el análisis del funcionamiento del vehículo y el seguimiento en tiempo real del mismo. Evitando que tenga que ir un ocupante en todo momento en el vehículo o esperar a que el vehículo se pare en algún punto para poder extraer dicha información.

Figura 1.2: Manejo del vehículos autónomos a distancia.

1.3 Prerequisitos

1.3.1 Pre-requisitos del sistema

Para el funcionamiento y funcionamiento del sistema será necesario dispones de:

- Disponer de un sistema operativo Linux/Windows/Mac sobre el que pueda ejecutar el interprete Python[5] 3.8 o superior.
- Librerías de Python especificas:
  - Django 3.1.4 ó +
  - mysqlclient 2.0.1 ó +
  - python-dateutil 2.8.1 ó +
  - parse 1.18.0 ó +
  - mysql-connector 2.2.9 ó +
  - django-mysql 3.9.0 ó +
  - pymysql 0.10.1 ó +
  - mysql-client 0.0.1 ó +
  - setuptools 50.3.2 ó +

[5]Python es un lenguaje de programación interpretado.

  - future 0.18.2 ó +
  - django-mathfilters 1.0.0 ó +
  - django-admin-list-filter-dropdown 1.0.3 ó +
  - openpyxl 3.0.5 ó +
  - channels 3.0.2 ó +
  - channels-redis 3.0.2 ó +
  - djangorestframework 3.12.2 ó +
  - django-createsuperuserwithpassword 1.0 ó +

- Base de datos MySQL con permisos de creación de tablas (o las tablas ya creadas) y permisos de lectura y escritura sobre la BBDD.

1.3.2 Otros Pre-Requisitos

Todo el desarrollo esta realizado sobre un entorno virtual Docker con el que se facilita la migración y la implantación entre sistemas operativos, y no sufre modificaciones de funcionamiento según el sistema que lo ejecute. También de esta forma permite su desarrollo durante la ejecución permitiendo resolver los errores al comienzo de proceso.

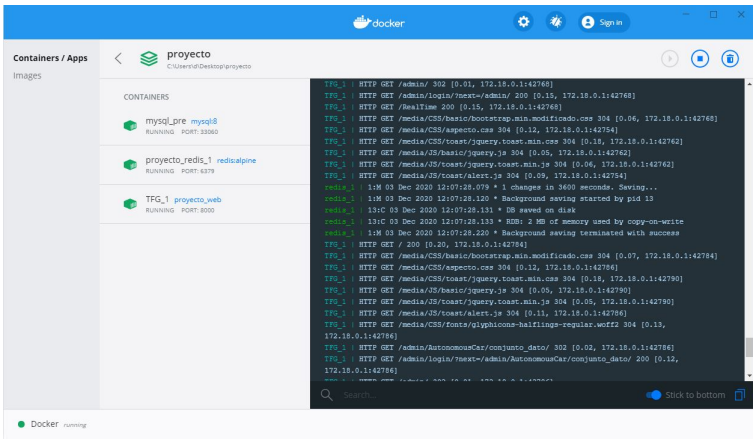

Figura 1.3: Proyecto Docker.

1.3.3 Compilación

Python es un lenguaje interpretado por lo que no necesita compilación, aunque si realiza un análisis del código con cada cambio que puede causar que la aplicación no se pueda ejecutar.

1.3.4 Estructura del documento

Este documento presenta la siguiente estructura:

- Portada del proyecto.
- Dedicatorias.
- Agradecimientos.

- Resumen del proyecto.
- Índice de contenidos, índice de figuras, índice de tablas e índices adicionales.
- Introducción: breve introducción al proyecto y al tema a tratar.
- Estados de arte: una introducción sobre las tecnologías utilizadas
- Desarrollo de la aplicación
- Capítulos del documento.
- Resultados de la solución
- Conclusiones y lineas futuras del proyecto.
- Apéndices con información de manuales de usuario e instalación y requerimientos para el desarrollo.
- Contraportada.

1.4 Motivación y objetivos

La motivación de este proyecto fue la posibilidad de aprender a desarrollar una web sobre un Framework con lenguaje Python (Django), así como perfeccionar mis conocimientos en Python. También existe la necesidad de un visualizador de datos de un vehículo autónomo, este proyecto además dará las prestaciones de poder hacerle seguimiento en tiempo real.

Los objetivos principales de este proyecto:

1. Primer objetivo: Registrar todos las datos provenientes de los sensores del sistemas ROS localizado en el vehículo.
2. Segundo objetivo: Visualizar en tiempo real el estado del vehículo.
3. Tercer objetivo: Poder simular viajes anteriores de los trayectos.
4. Cuarto objetivo: Poder ver resultado estadísticos de los trayectos.
5. Quinto objetivo: Poder extraer datos en Excel o csv de los datos de un trayecto.

Capítulo 2

Estudio teórico

...Encontré mil y una palabras que decir, pero esta singularidad de irracionalidad en la que vivimos, nos hizo cuerdos a los locos para enseñarnos que un mundo de ciegos el tuerto es el rey...

Anónimo

2.1 Introducción

En este capítulo se va a tratar todos los aspectos del estudio teórico y previo al desarrollo del proyecto.

El capítulo se estructura en los siguientes apartados:

1. Estado del Arte: define las tecnologías existentes a utilizar en el proyecto.
2. Técnicas utilizadas: define las metodologías a seguir para el desarrollo del proyecto.
3. Conclusiones

2.2 Estado del Arte

2.2.1 Python-Django

El lenguaje Python es un tipo de lenguaje interpretado, que esta ahora en auge, en el que tiene una estructura que facilita la lectura del código. Es un lenguaje que soporta programación orientada a objecto, programación imperativa y programación funcional (aunque en menor medida). Django es un Framework de código abierto con el que somos capaces de hacer desarrollos web mediante el lenguaje de programación Python. Se basa en el principio de MVC (Modelo-Vista-Controlador) aunque lo definen de manera diferente MVT (Model-View-Template):

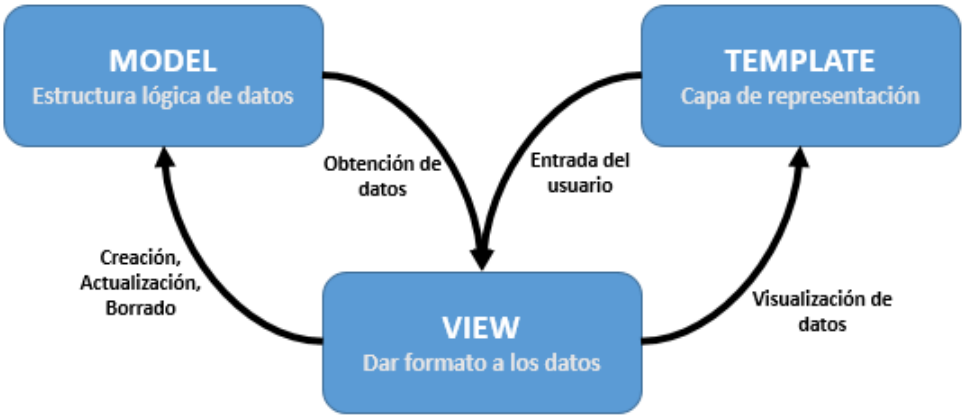

Figura 2.1: Diagrama de modelo MVT.

2.2.2 MySQL

Es un sistema de base de datos relacional desarrollado por *Oracle Corporation*. Es uno de los gestores más utilizados al ser gratuito, siendo muy fácil de usar, bastante rápida, diferentes capas de seguridad y capaz de funcionar con pocos requerimiento y utilizando su memoria de manera eficaz. Además tiene flexibilidad con Sistemas Linux y Windows. Fue escogida debido a estas cualidades y a su comportamiento eficiente en el trabajo de procesamiento de transacciones y su capacidad de consulta.

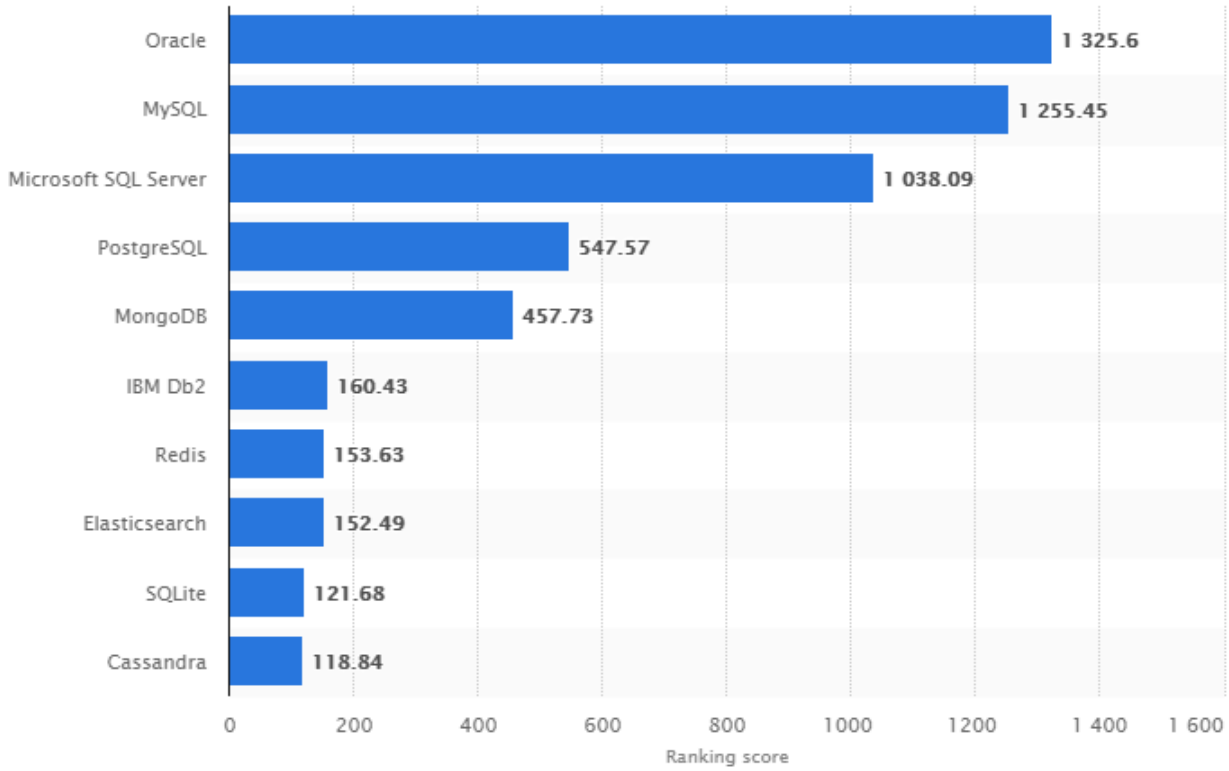

Figura 2.2: Estadísticas de popularización de base de datos sacadas de la página www.statista.com en diciembre 2020.

2.2.3 Docker

Es una aplicación de código abierto que permite automatizar despliegues mediante contenedores de aplicaciones de manera independiente generando una capa de abstracción similar a una virtualización aprovechando la base de un sistema operativo único y permitiendo con facilidad diferentes versiones dentro del mismo sistema. Gracias a este aplicativo se permite un despliegue rápido e independiente.

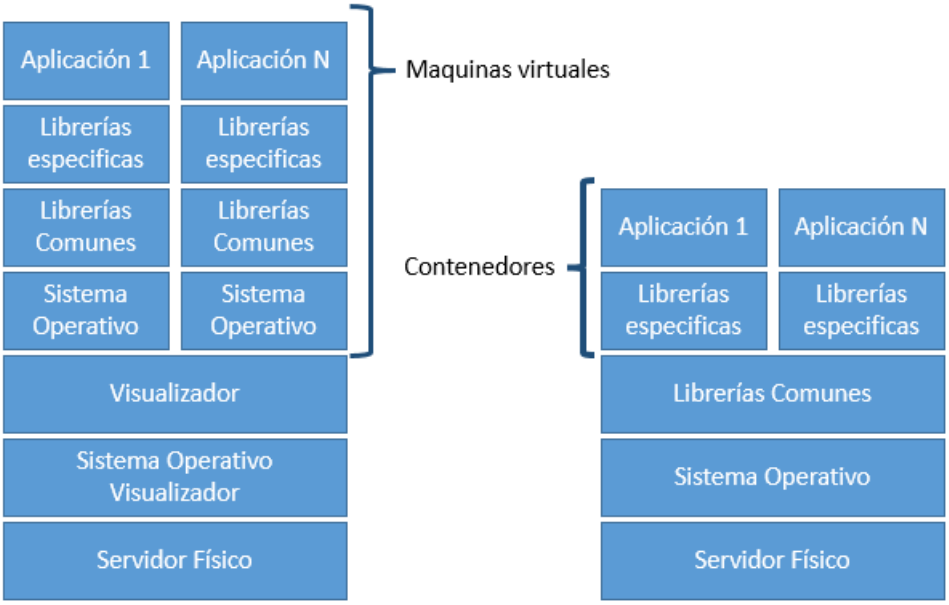

Figura 2.3: Diagrama comparativo máquinas virtuales y contenedores.

2.2.4 Gráficas HTML

Para la creación de gráficas en HTML se usará la librería Chart.js de software libre, para poder realizar visualizaciones de los datos de manera gráfica sin demasiada complejidad. Para ello se trabajará con dos versiones combinando ambas en una sola:

- Versión 2.8: Versión estable del proceso, con muchas visualizaciones diferentes como:
  - Gráficas de barras
  - Gráficas de línea
  - Gráficas de área
  - Gráficas de rosquilla
  - Gráficas de logarítmicas
- Versión 2.6.4: Versión antigua en la cual se podía visualizar de mejor manera las gráficas polares siendo capaz de representan en 2D de manera correcta sobre coordenadas cartesianas que en la versión 2.8 presenta fallos con los valores negativos.

Desde su web Chart.js se puede aprender con facilidad a utilizar dicha librería, además de contar con ejemplos y guías de aprendizaje.

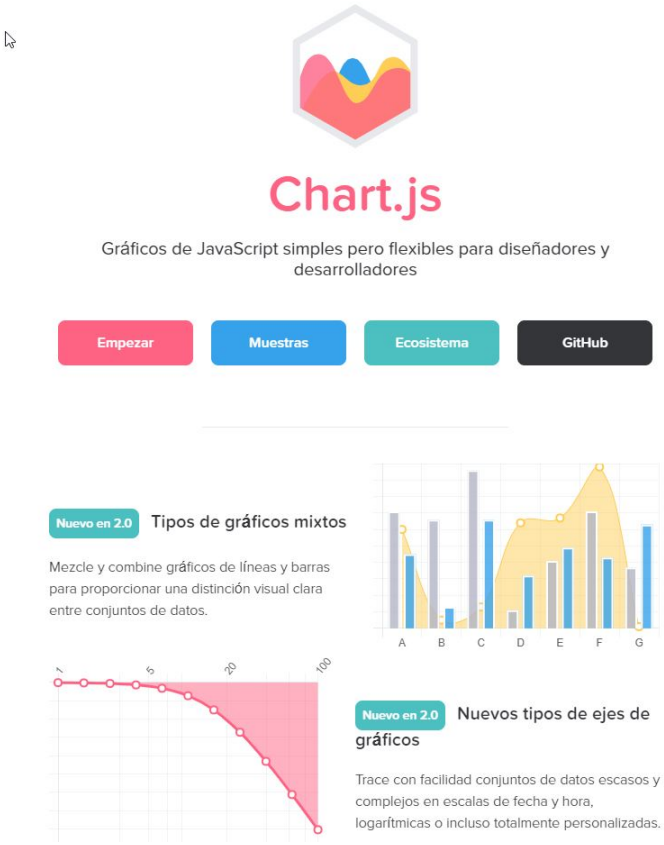

Figura 2.4: Página web Chart.js.

2.2.5 Canales Redis

Redis es un motor de base de datos en memoria que se utiliza como memoria temporal o memoria pasarela, aunque también se puede utilizar como base de datos persistente. Los canales Redis se utilizan en Django como canales de comunicación con la parte del backend para permitir tener conexiones abiertas entre el navegador y el servidor, y permitir respuestas en tiempo real, evitando la saturación de llamadas del servidor.

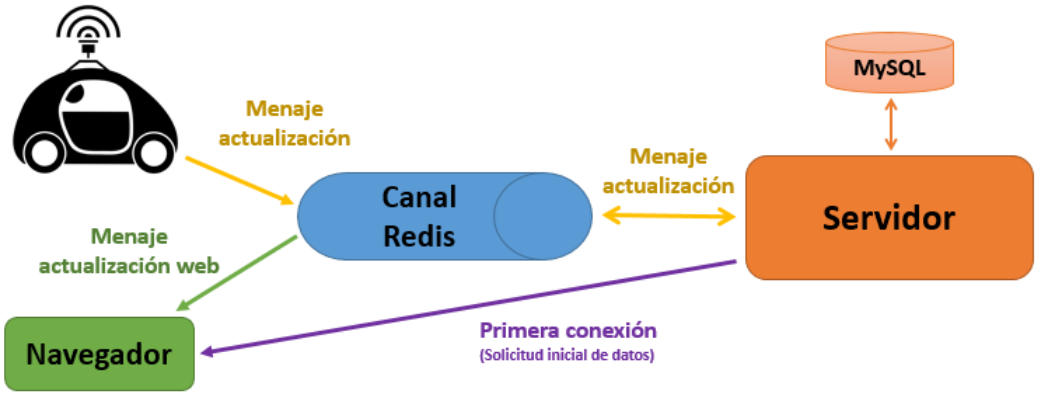

Figura 2.5: Diagrama de comunicación sobre un canal Redis.

2.3 Técnicas utilizadas

2.3.1 Metodología Agile

Se decide tomar una iniciativa de desarrollo ágil que facilite ver la estabilidad del sistema e ir realizando pruebas de los desarrollos. Permite realizar pequeñas modificaciones que se adaptan a la idea final.

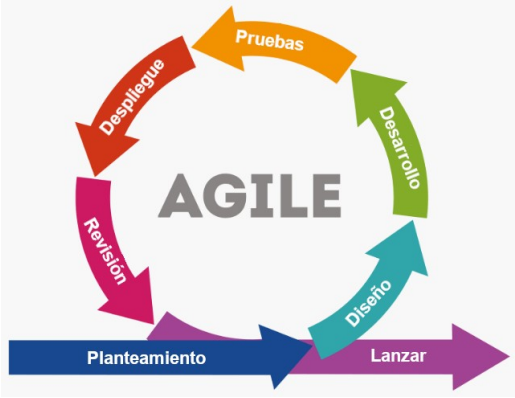

Figura 2.6: Diagrama Metodología Agile.

2.3.2 Elementos Modularizables o Plugins

La mayor bondad de toda aplicación es su fácil incorporar elementos que se puedan añadir y quitar sin que afecte al resto del sistema. Por eso cada representación se realiza en módulos permitiendo que se puedan representar cada una de las variables de manera independiente.

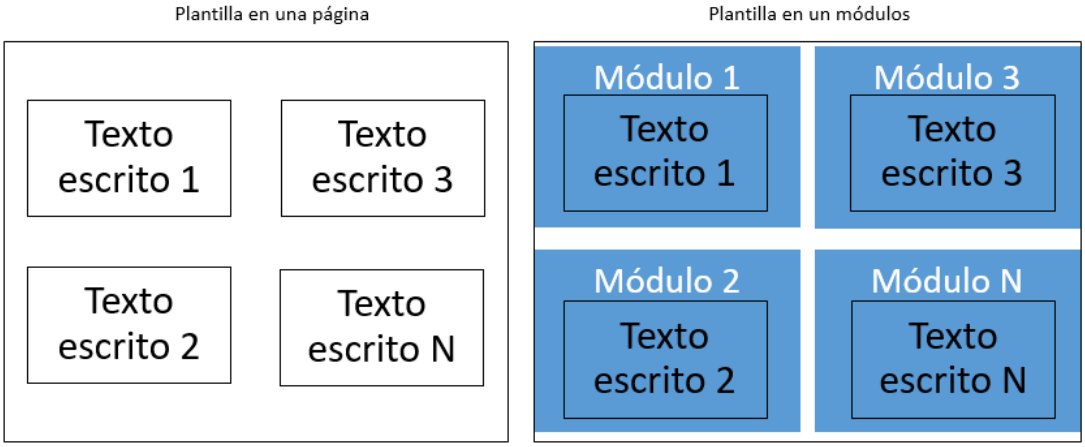

Figura 2.7: Comparativa desarrollo web modular.

Cada módulo se desarrolla como una página independiente permitiendo crear, eliminar o editar cada una de las partes sin interferir en el resto. Cada una de las dependencias de cada módulo se añaden en ese módulo (CSS, JavaScripts, etc.).

2.4 Conclusiones

La combinación de todas estas técnicas y tecnologías serán la parte fundamental del desarrollo de este proyecto.

Capítulo 3

Desarrollo

La vida es muy simple pero insistimos en hacerla complicada.

Confucio

3.1 Introducción

Al seguir una metodología Agile, se puede dividir el desarrollo en fases:

- Fase 1: Preparación del sistema
- Fase 2: Desarrollo de la página principal.
- Fase 3: Desarrollo de módulos.
- Fase 4: Desarrollo de backend y estructura base de datos.
- Fase 5: Desarrollo de viajes.
- Fase 6: Desarrollo de páginas de estadísticas.
- Fase 7: Desarrollo de otras páginas.
- Fase 8: Desarrollo de servicio para cargar de datos.
- Fase 9: Mejora de rendimiento y comunicación web mediante canales.
- Fase 10: Adaptación del desarrollo al simulador Carla.

3.2 Preparación del sistema

Los primeros pasos es instalar el motor Python 3.8, es sobre el cual se ejecutará el proceso. También son necesarias una serie de librerías básicas:

1. pip install Django
2. pip install mysqlclient

3. pip install mysql-connector

4. pip install django-mysql

Con estas premisas ya podemos iniciar el desarrollo, para ejecutar un proyecto Django ejecutamos el comando con nombre de proyecto *AutonomousCar*:

django-admin startproject AutonomousCar

3.2.1 Configuración de rutas fijas

En el fichero *TFG/TFG/settings.py* se realizaran las configuraciones necesarias, como el idioma, la zona, apps, puerto, etc. Pero cabe destacar la configuración de las rutas de principales de los ficheros estáticos:

Listado 3.1: Configuración de la ruta para objectos fijos de la web

STATICFILES_DIRS = (os.path.join("TFG",.^AutonomousCar/media"),)

3.3 Desarrollo página principal

El desarrollo se comienza con la creación de la primera página web y para ello es necesario realizar una serie de configuraciones previas en Django. En el fichero *TFG/AutonomousCar/view.py* se crea la función a la cual realizamos la generación de la página web.

Listado 3.2: Código página pruebas view.py

```
1 from django.http import HttpResponse
2 def index(request):
3     template_base = get_template(index.html")
4     dato = {"Velocidad":20.0, "EjeX":20.0, "EjeY":10.0, "Giro":-30.0,
5         "Senial":70,"Semaforo":"rojo"}
6     pagina = template_base.render(dato)
7     return HttpResponse(pagina)
```

Para poder hacer la carga ese fichero index.html que se encuentra en una de las rutas estáticas deberá ser almacenado en el grupo de Urls accesibles por el sistema en el fichero de la ruta *TFG/TFG/urls.py* es necesario identificarla en como un *urlpatterns*:

Listado 3.3: Código referencia página pruebas en urls.py

```
from django.conf.urls import url
from django.contrib import admin
from index import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index)
]
```

El diseño de la web index.html, representara los valores de los datos, en este caso los muestra como un único bloque de texto, como se puede ver en la imagen, podremos verificar el funcionamiento de la web y comenzar a construir los diferente módulos específicos:

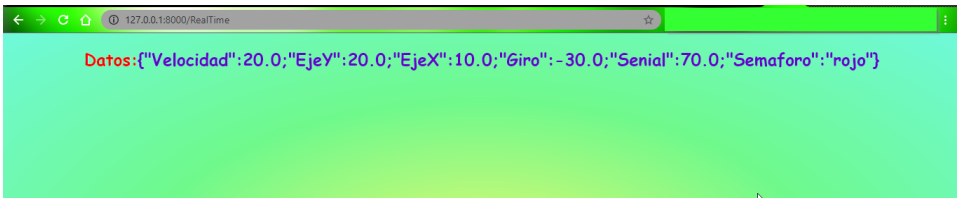

Figura 3.1: Primera prueba de funcionamiento.

3.3.1 Carga de datos por modelo y funciones básicas

Para poder representar los datos que se envían desde el servidor y poder construir a partir de ellos una estructura de datos, es necesario indicar en Django el valor al que hace referencia dentro del propio código de HTML:

Listado 3.4: Parte 1 del código del template del módulo de posición

```
<div class='izquierda'>
    {% if dato.PosicionX != undefined and dato.PosicionY != undefined
      and dato.PosicionAltura != undefined %}
        <p><b>Posición X:</b><span id='PosicionX' class='transiciones_globales'>
            {{dato.PosicionX}}</span><input id='PosicionXcrear' type='number'
            class='input_oculto' value=0></p>
        <p><b>Posición Y:</b><span id='PosicionY' class='transiciones_globales'>
            {{dato.PosicionY}}</span><input id='PosicionYcrear' type='number'
            class='input_oculto' value=0></p>
        <p><b>Altura:</b><span id='PosicionAltura' class='transiciones_globales'>
            {{dato.PosicionAltura}}</span><input id='PosicionAlturacrear'
            type='number' class='input_oculto' value=0></p>
    {% elif dato.x != undefined and dato.y != undefined and dato.z != undefined %}
        <p><b>Posición X:</b><span id='PosicionX' class='transiciones_globales'>{{dato.x}}
            </span><input id='PosicionXcrear' type='number' class='input_oculto' value=0></p>
        <p><b>Posición Y:</b><span id='PosicionY' class='transiciones_globales'>{{dato.y}}
            </span><input id='PosicionYcrear' type='number' class='input_oculto' value=0></p>
        <p><b>Altura:</b><span id='PosicionAltura' class='transiciones_globales'>{{dato.z}}
            </span><input id='PosicionAlturacrear' type='number' class='input_oculto'
             value=0></p>
    {% else %}
        <p><b>Posición X:</b><input id='PosicionXcrear' type='number' value=0></p>
        <p><b>Posición Y:</b><input id='PosicionYcrear' type='number' value=0></p>
        <p><b>Altura:</b><input id='PosicionAlturacrear' type='number'
            value=0></p>
    {% endif %}
</div>
```

En azul podemos ver como representa el Valor de un dato y en rojo las funciones básicas de comparación. Las funciones de comparación pueden ser:

- Las funciones igual que: {% if VARIABLE == "a" %}
- Las funciones distinto que: {% if VARIABLE != "b" %}
- Las funciones mayor que : {% if VARIABLE >5 %}
- Las funciones menor que : {% if VARIABLE <5 %}
- Las funciones mayor o igual que : {% if VARIABLE >= 5 %}
- Las funciones menor o igual que : {% if VARIABLE <= 5 %}

Siempre debe acabar con {% endif %}, y pueden existir {% else %} y también funciones anidadas {% elif VARIABLE <= 5 %}. Esta funciones generan el HTML de la web en función del resultado.

3.3.2 Bucle For

Otras de las funciones utilizadas en el proyecto son los Bucle For para la creación de los datos en tablas o diferentes conjuntos de datos:

Listado 3.5: Parte 2 del código del template del módulo de posición

```
<tbody id='DatosTabla'>
    {% for posicion in dato.posiciones %}
    <tr>
        <td>{{forloop.counter}}</td>
        {% if posicion.PosicionX != undefined and posicion.PosicionY != undefined and
                posicion.PosicionAltura != undefined %}
            <td>{{posicion.PosicionX}}</td>
            <td>{{posicion.PosicionY}}</td>
            <td>{{posicion.PosicionAltura}}</td>
        {% elif posicion.x != undefined and posicion.y != undefined and posicion.z != undefined %}
            <td>{{posicion.x}}</td>
            <td>{{posicion.y}}</td>
            <td>{{posicion.z}}</td>
        {% endif %}
    </tr>
    {% endfor %}
</tbody>
```

El Bucle For también esta compuesto de una apertura y un cierre haciendo una similitud al funcionamiento de un bucle en Python recorriendo elementos extrayendo sus datos y pudiendo hacer referencia a un enumerador de cada posición con un contador del bucle

3.3.3 Otros Tags

En la página de documentación de Django-Template-Language podemos ver la lista el lenguaje completo que se puede utilizar en los templates[1] para construir el código HTML. En el apartado 3.4 se mostrar cómo se inserta un template en otro utilizando otro tipos de Tags.

3.3.4 Creación de filtros propios o TAGS

Otra funcionabilidad útil es la creación de tus propios filtros, funciones o tags y así poder interactuar con los valores del modelo o para introducir otro tipo de valores. Para implementar esta funcionabilidad es importar crear:

- Carpeta *templatetags* en la raiz del proyecto
- Crear dentro de esta carpeta un fichero vacio: *___init___.py*

[1] Templates: plantillas Django para mostrar los datos del modelo.

- Un fichero donde crearemos la siguiente funciones.*Por ejemplo str_round.py*, dentro del fichero:

Listado 3.6: Código fuente del fichero str_round.py

```
from django import template

register = template.Library()

@register.filter
def str_round_value(value,arg):
    valor=""
    try:
        valor=round(value, arg)
    except:
        try:
           valor=round(float(value), int(arg))
        except:
            None
    if valor == "":
        valor = 0
    return valor
@register.filter
def str_round(value):
    return str_round_value(value,2)
```

Y desde el código HTML cargamos con el operador load el fichero *.py y llamamos a la función:

Listado 3.7: Código de ejemplo de uso la librería str_round.py en un template

```
<div>
    {% load str_round %}
     VALOR|default:4.2568
    {{ % VALOR | str_round %}} = 4.26
    {{ % VALOR | str_round_value:3 %}} = 4.257
    {{ % VALOR | str_round_value:3 | str_round %}} = 4.26
</div>
```

Se permiten encolar tantas funciones como se desee, pero el orden siempre es una tras otra hacia la derecha.
Durante el proceso se hace uso también de la librería *django-mathfilters* que facilita el uso de algunas de estas funcionabilidades. Existen otro tipo de TAG que se pueden crear que se pueden encontrar en la documentación de Django-tag, en el que describe bastante claro todas las funcionabilidades aunque recomiendo disponer de una página en la que ir realizando pruebas de cada módulo implementado.

3.4 Desarrollo de módulos

Tras la comprobación del funcionamiento podemos modificar este index.html para que a su vez llame a otras páginas o módulos, permitiendo hacer una creación estructurada y sin dependencia entre módulos, solo en los datos que se envían desde el servidor. Se crean diferente módulos:

Listado 3.8: Código desarrollo modular

```
<html>
    {% include “ElementosFijos/Cabeceras.html” %}
    <body>
        <header style=“margin:0px;padding:0px;” >
            {% include “ElementosFijos/Menu.html” %}
        </header>
        <section>
            {% include “Visualizador/VisualizadorDatos.html” %}
        </section>
        <footer>
            {% include “ElementosFijos/PiePagina.html” %}
        </footer>
    </body>
</html>
```

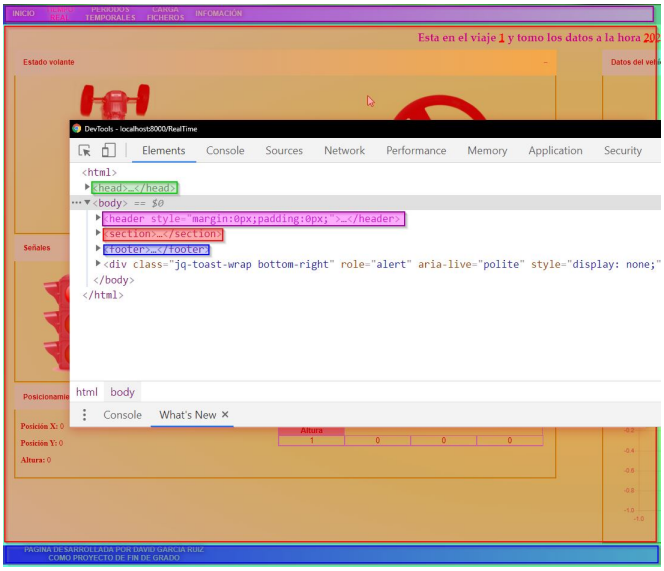

Figura 3.2: Detalles de diferente módulos HTML que contiene a su vez otra parte de página HTML.

La estructura de la página web se basa en:

- El *head* esta formado por la librerías Javascript y he información del web, tiene valores que dependen de un parámetro que se envía para cambiar el titulo de la página.
- El *header* hace referencia a los menús superiores de la pagina, los cuales se componen a su vez de dos partes, menú superior izquierdo con las páginas básicas y menú superior derecho con las páginas de la sección de administración de Django desde la cual se puede analizar los datos que se cargaron con anterioridad.
- El *footer* es la barra inferior de la pagina web, hace referencia solo al autor de la web y al nombre de la aplicación.
- El *seccion* esta formado por pequeños módulos para mostrar los datos de manera independiente,

con sus funciones y adjuntos(librerías, imágenes, etc.). Cada uno de estos módulos son descritos a lo largo de los siguientes puntos.

3.4.1 Estado del volante

Los datos sobre el giro del volante se basan en una estructura de datos que en función de un valor positivo o negativo genera una rotación de las imágenes representativas de las ruedas generando una inclinación de ellas respecto del eje del vehículo y una rotación sobre el volante mediante un función Javascript y CSS. Dichos valor se representa en el centro del volante para conocer su valor real.

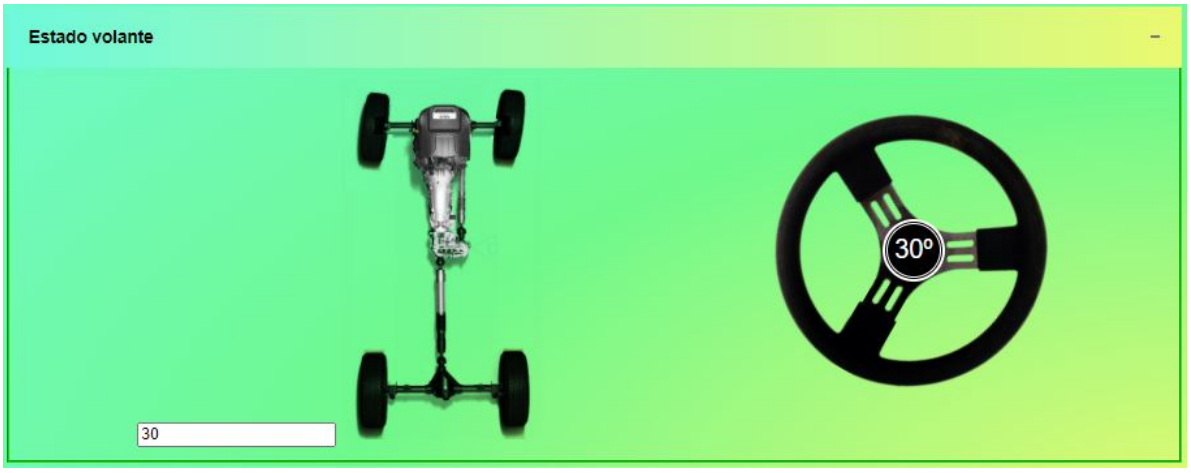

Figura 3.3: Representación giro del volante 30º

Indicando mediante script que detecte el cambio de valor en el *input* de giro del volante se aplicará una rotación de las imágenes de las ruedas y del volante, similar al del recuadro:

Listado 3.9: Código JavaScript para giro de ruedas.

```
self.parent().parent().find(".giro_ruedas").css({
  "-webkit-transform" :"rotate("+giro_ruedas+"deg)",
  "-webkit-transform" :"rotate("+giro_ruedas+"deg)",
  "-moz-transform" :"rotate("+giro_ruedas+"deg)",
  "-o-transform" :"rotate("+giro_ruedas+"deg)"
});
```

3.4.2 Datos del vehículo

Los datos sobre la velocidad del vehículo y la marcha se pueden visualizar sobre un velocímetro, el cual indica la velocidad de forma número y visual, con límite fijado en 100km/h, y un marcador con forma de display que indica la marcha (R, H o del 1 al 6).

Figura 3.4: Velocímetro y display de marcha

Utilizando un código similar al aplicado en las imágenes de giro de ruedas se realiza un giro sobre la aguja del velocímetro, para hacer el cálculo de la rotación se debe tener en cuenta la posición inicial y posición final(o posición máxima de la aguja). Contando que el 30 % del giro de la aguja esta fuera de rango, debes realizar el cálculo:

Listado 3.10: Funciones para cálculo del angulo de la aguja del velocímetro.

```
Posicion_Original = -60
Angulo_Aguja = Posicion_Original + Velocidad * (330 / Velocidad_Max)
```

Con ese valor de Angulo_Aguja se determina el giro de la aguja y la posición de la punta de la misma, indicando que marcadores del velocímetro (separadores y números de la velocidad) deben iluminarse con mayor luminosidad para mantener un aspecto visual más atractivo.

También se hace una representación de la velocidad con valor numérico en el centro del velocímetro para poder analizar con mayor precisión la velocidad exacta del vehículo.

Para la representación del display de la machar del vehículo, se utiliza la lógica similar de un *descodificador BCD*[2]. Aplicando esa lógica mediante Javascript se realizarán las representaciones de del valor numérico ocultando las diferentes barras del display o cambiando el color en el caso de indicar marcha atrás.

[2]Descodificador BCD: mediante 4 bits es capaz de generar valores en binarios diferentes pines, para poder hacer representaciones en un *display led* de 7 pines

3.4.3 Señales

Según el dato recibido sobre el último estado del semáforo se mostrará una imagen del semáforo en rojo, verde o amarillo. También en función del valor muestra la señal de velocidad, indicando valores positivos señales de velocidad recomendada y señales de límite de velocidad para valores negativos (como se muestra el valor de -30).

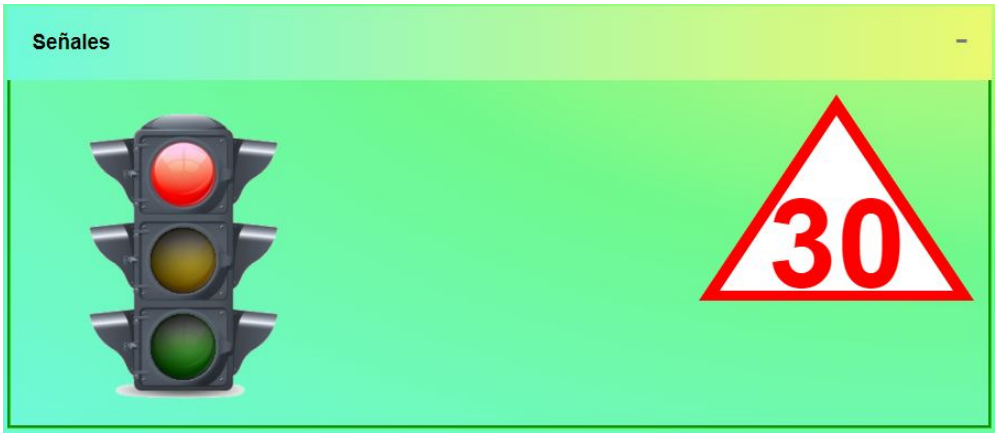

Figura 3.5: Semáforo rojo y última señal valor velocidad máxima permitida

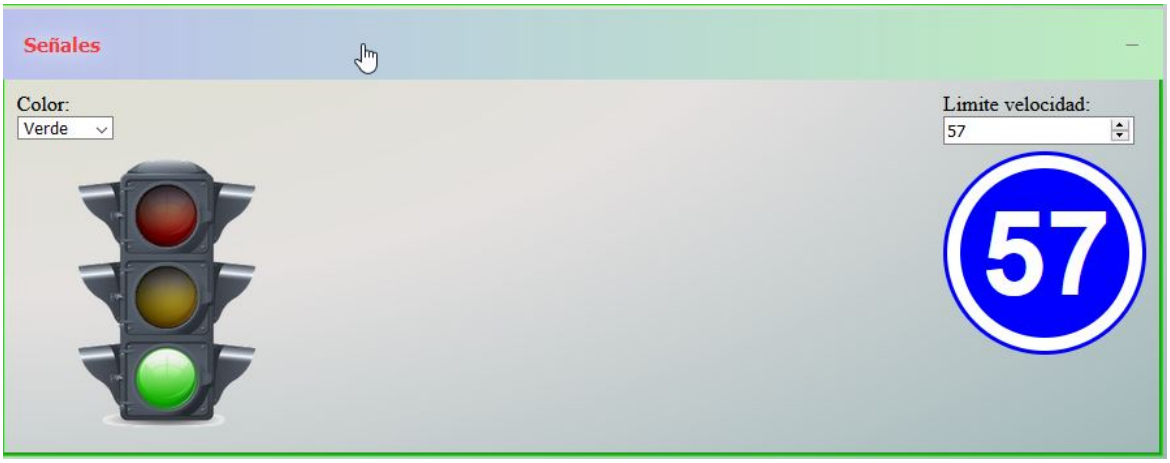

Figura 3.6: Semáforo verde y última señal velocidad recomendada

Para poder realizar este cambio entre señales se usa la misma estructura y de objecto en capas y se modifica solamente mediante CSS el aspecto de la señal.

CSS para las señales de velocidad recomendada con forma circular:

Listado 3.11: Código CSS para aspecto de señal de velocidad recomendada.

```
div#tipo_senial.circulo {
    margin: 0px;
    padding: 0px;
    padding-top: 10px;
    width: 150px;
    height: 140px;
    -moz-border-radius: 50
    -webkit-border-radius: 50
    border-radius: 50
    background: white;
    border: 3px solid blue;
    text-align: center;
    vertical-align: middle;
}
div#tipo_senial.circulo div {
    margin: 0px;
    width: 130px;
    height: 130px;
    margin-left: 10px;
    -moz-border-radius: 50
    -webkit-border-radius: 50
    border-radius: 50
    background: blue;
}
div#tipo_senial.circulo p.threedigitos {
    margin: 0px;
    margin-top: -105px;
    height: 130px;
    width: 130px;
    margin-left: 11px;
    color: white;
    font-family: arial;
    font-size: 70px;
    font-weight: bold;
}
div#tipo_senial.circulo p.twodigitos {
    margin: 0px;
    margin-top: -115px;
    height: 130px;
    width: 130px;
    margin-left: 11px;
    color: white;
    font-family: arial;
    font-size: 90px;
    font-weight: bold;
}
```

El CSS para las señales de límite de velocidad con forma de triángulo:

Listado 3.12: Código CSS para aspecto de señal de prohibición.

```
div#tipo_senial.triangulo div {
    width: 0;
    margin: 0px;
    height: 0;
    border-right: 85px solid transparent;
    border-top: 15px solid transparent;
    border-left: 85px solid transparent;
    border-bottom: 127.5px solid white;
    margin-left: -85px;
}

div#tipo_senial.triangulo p.threedigitos {
    margin: 0px;
    margin-top: -67px;
    width: 200px;
    margin-left: -100px;
    color: red;
    font-family: arial;
    font-size: 70px;
    font-weight: bold;
}

div#tipo_senial.triangulo p.twodigitos {
    margin: 0px;
    margin-top: -85px;
    width: 200px;
    margin-left: -100px;
    color: red;
    font-family: arial;
    font-size: 90px;
    font-weight: bold;
}
```

3.4.4 Posicionamiento y Mapa

Mediante una tabla podemos visualizar la posiciones GPS[3] en las que estuvo el vehículo y el estado actual. Con el mapa podemos visualizar como fue el recorrido del vehículo mediante una gráfica en 2D.

Figura 3.7: Tabla de posiciones GPS y representación en plano 2D

En la imagen se pueden visualizar las 3 posiciones actuales y una tabla la cual reflejará todos los puntos del viaje en orden inverso por los que estuvo el vehículo. En el mapa se puede visualizar el recorrido realizado por el vehículo. Se definirá su funcionamiento en la parte de estadistas, mapa cartesiano en el punto 3.7.1.5.

3.4.5 Otros datos del sistema real

Existen otra serie de campos existentes en los ficheros devueltos por el sistema ROS del coche. Los campos extra son:

- Distancia entre ejes (wheelbase): parámetro del vehículo, distancia entre ejes de ruedas.
- Aceleración (x_acc): aceleración del vehículo.
- Velocidad en Z (z_speed): velocidad del coche en coordenada Z.
- Orientación (yaw): orientación del vehículo.
- Cabeceo del vehículo (yaw_ref): referencia de cabeceo a la que se tiene que adecuar el vehículo.

[3]GPS (Global Positioning System): Sistema de posicionamiento global en 3 dimensiones.

- Estado del acelerador (throttle): posición de acelerador.
- Estado del freno (brake): posición de freno.
- Angulo pitch (pitch): ángulo de pitch del vehículo.
- Angulo roll (roll): ángulo de roll del vehículo.

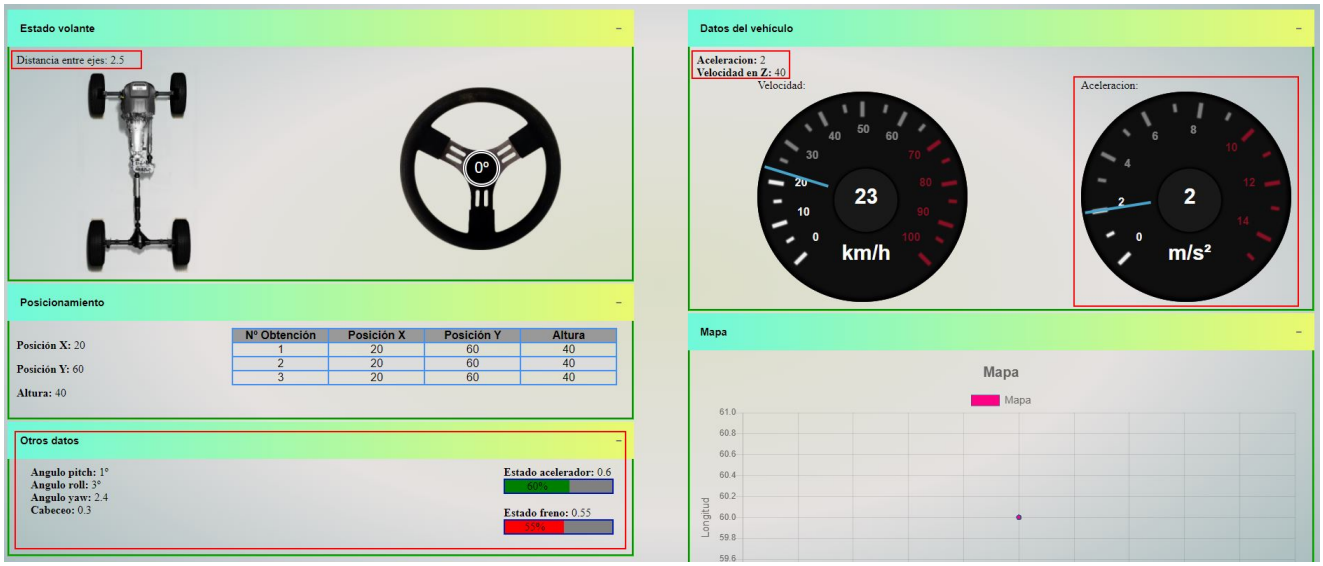

Figura 3.8: Imagen en la que se resaltan los otros campos existente en un viaje real.

Las barras de acelerador y freno marcan el estado del mismo en función del 1 o 0 indicando el porcentaje fuerza de pisado del pedal.
Según la distancia de los eje, un valor que solo se refleja en el segundo modelo de datos del vehículo se modifica la distancia de los ejes mediante CSS, con valores de 0 a -25 %, ejemplo:

Listado 3.13: Formulas para ajustar distancia entre ejes.

```
distancia = 4 - distancia_ejes_real
CSS: { margin-top:-distancia%; }
```

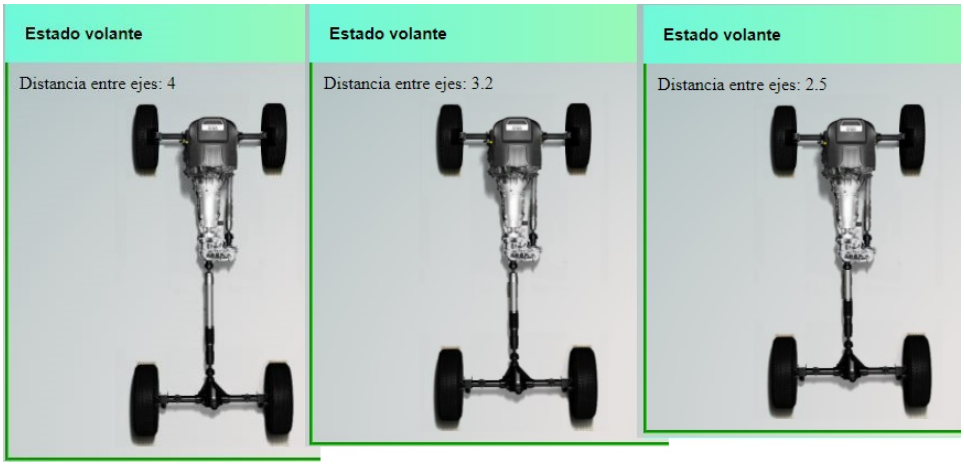

Figura 3.9: Comparativa de distancia con 4, 3.2 y 2.5 .

3.5 Desarrollo de backend y estructura base de datos

Parte del backend en la que se maneja los datos y la estructura de esos datos para poder tratarlos es necesario una conexión con la base de datos, la creación de un modelo de relación con esa estructura de datos para su posterior manejo en los templates como valores, poder generar ficheros con dichas estructuras de datos y para ser visualizados durante las comunicación de datos en las pantallas de visualización cuando se soliciten al servidor.

3.5.1 Estructura de datos

La lista de campos creados para el visualizador son:

Tabla 3.1: Definición campos del proyecto.

Campo	***Tipo***	***Rango***
Id	Numérico	De 1 a ∞
Viaje	Numérico	De 1 a ∞
Tiempo	Fecha	Fecha y hora
Velocidad	Numérico	De 0 a ∞
PosicionY	Numérico	Positivo y negativo
PosicionX	Numérico	Positivo y negativo
PosicionAltura	Numérico	Positivo y negativo
Marcha	Numérico	Del -1 al 6
AnguloGiro	Numérico	Positivo y negativo
SenialLimite(Señal)	Numérico	Positivo y negativo
Semaforo	Texto	Valor “rojo”, “amarillo” y “verde”
Posiciones*	Array	Array de datos de posición X, Y, Altura

**Existen campos adicionales que se envían en cada petición de datos pero no son campos almacenados en la base de datos, como por ejemplo el conjunto de posiciones.*

3.5.2 Creación de tabla en base de datos

Para la creación de los campos en la base de datos son necesarias las librerías de MySQL. Para la creación de tablas en Django es relativamente sencillo realizar dichas creaciones, para ello se debe implementar modelos. Dichos modelos hacen referencia a cada uno de los campos de los que este compuesta una clase de tipo *models.Model*.

- Es necesario agregar en el fichero de “settings.py” en el apartado de DATABASES la configuración de la base de datos a la que se van a conectar los modelos (este proyecto MySQL):

Listado 3.14: Configuración del MySQL en Django.

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.mysql",
        "NAME": "Visualizador",
        "USER":"user",
        "PASSWORD":"password",
        "HOST":"mysql_pre",
        "PORT":"3306"
    }
}
```

- Existen diferentes tipos de datos para introducir en las clases de los modelos:
  - CharField: se usa para definir cadenas de longitud corta a media. Se debe especificar la longitud máxima (max_length) de los datos que se guardarán.
  - TextField: se usa para cadenas de longitud grande o arbitraria. Puedes especificar una longitud determinada (max_length) para el campo, pero sólo se usa cuando el campo se muestra en formularios (no se fija ese valor a nivel de la base de datos).
  - IntegerField: es un campo para almacenar valores de números enteros y para validar los valores introducidos como enteros en los formularios.
  - BooleanField:se usa para valores booleando de verdadero o falso.
  - FloatField: es un campo para almacenar valores de números de tipo float y para validar los valores introducidos como enteros en los formularios.
  - DateField y DateTimeField; se usan para guarda y representar fechas e información fecha y hora (como en los objetos Python datetime.date y datetime.datetime). Estos campos pueden adicionalmente declararse con los parámetros auto_now=True (para establecer el campo a la fecha actual cada vez que se guarda el modelo) o auto_now_add (para establecer sólo la fecha cuando se crea el modelo por primera vez). También se puede fijar un valor por defecto (default) en lugar de los otros dos parámetros,para establecer una fecha por defecto que puede ser sobrescrita por el usuario.
  - EmailField: se usa para validar direcciones de correo electrónico. Siendo un campo de tipo TextField.
  - FileField(FieldFile) e ImageField: se usan para subir ficheros e imágenes respectivamente (el ImageField añade simplemente una validación adicional de que el fichero subido es una imagen). Estos tipos de datos tienen parámetros para definir cómo y donde se guardan los ficheros subidos. También existe el tipo *FilePathField* en el que se guarda la ubicación del fichero en servidor.
  - GenericIPAddressField:se usa para validar direcciones IP de tipo IPv4 o IPv6. Siendo un campo de tipo CharField.
  - AutoField: es un tipo especial de IntegerField, se incrementa automáticamente. Cuando no específicas una clave primaria para tu modelo, se añade automáticamente un campo de este tipo.
  - OneToOneField: se usa para especificar una relación uno a uno con otro modelo de la base de datos (ej. un número de DNI de una persona y un número de permiso de conducir).

  - ForeignKey: se usa para especificar una relación uno a muchos con otro modelo de la base de datos (ej. un coche tiene un fabricante, pero un fabricante puede hacer muchos coches). El lado üno"de la relación es el modelo que contiene la clave.
  - ManyToManyField: se usa para especificar una relación muchos a muchos (ej. un libro puede tener varios géneros, y cada género puede contener varios libros).
  - Existen otros muchos tipos de modelos, muchos más específicos la lista completo se puede encontrar en: djangoproject:field-types

- Agregar en el fichero de "models.py" una clase dependiente 'models.Model' para que se pueda automontar, con validadores, seleccionables permitidos (encaso de semáforo) y valores por defecto: (Es necesario en el fichero de "admin.py" incluir el modelo creado)

Listado 3.15: Modelo del visualizador.

```
class conjunto_dato(models.Model):
  id = models.AutoField(primary_key=True, unique=True,auto_created=True, verbose_name="
      Identificador")
  Tiempo = models.DateTimeField(verbose_name="Momento captura",auto_now_add=True)
  Viaje = models.IntegerField(verbose_name="Identificador de trayecto", validators=[
      validar_viaje], default=1)
  AnguloGiro = models.IntegerField(verbose_name="Angulo de Giro", validators=[validar_angulo],
       default=0)
  Velocidad = models.IntegerField(verbose_name="Velocidad", validators=[validar_velocidad],
      default=0)
  Marcha = models.IntegerField(verbose_name="Marcha", validators=[validar_marcha], default=0)
  Semaforo = models.CharField(max_length=9, verbose_name"Ultimo semaforo", choices=
      conjunto_semaforo, default="rojo")
  SenialLimite = models.IntegerField(verbose_name="Senial de limite", default=-30)
  PosicionX = models.IntegerField(verbose_name="Posicion X", default=0)
  PosicionY = models.IntegerField(verbose_name="Posicion Y", default=0)
  PosicionAltura = models.IntegerField(verbose_name="Posicion Altura", default=0)
```

- Para que se auto creen las tablas será necesario ejecutar los comandos:
  - Creación de la migración a la base de datos según fichero "model.py":
    - python proyecto/manage.py makemigrations
  - Realiza los cambios de la migración creada:
    - python proyecto/manage.py migrate
  - Creación de un usuario para logarse desde la parte de admin y poder visualizar la base de datos desde la parte backend de Django:
    - python proyecto/manage.py createsuperuserwithpassword −−username user −−password pass −−email root@root.com −−preserve

3.5.3 Estructura de datos del vehículo real o simulador Carla

Al no compartir los mismos valores no se almacena en la misma tabla de la base de datos, la estructura de los campos de los ficheros con datos reales recopilados por el sistema ROS del vehículo o provenientes del simulador Carla[4] son:

Tabla 3.2: Definición campos del simulador Carla.

Campo	***Tipo***	***Rango***	***Definición***
brake	Numérico	De 0 a 1	Representa el estado del freno
yaw_ref	Numérico	De -∞ a ∞	Indica el ángulo de cabeceo del vehículo
pitch	Numérico	De -∞ a ∞	Ángulo de pitch del vehículo
deltaseconds	Numérico	De 0 a ∞	Indica la marca temporal entre un dato y otro
throttle	Numérico	De 0 a 1	Representa el estado del acelerador
roll	Numérico	De -∞ a ∞	Ángulo de roll del vehículo
lon_speed	Numérico	De 0 a ∞	Indica la velocidad del vehículo
x_acc	Numérico	De -∞ a ∞	Indica la aceleración del vehículo en el eje X
z_speed	Numérico	De -∞ a ∞	Indica la velocidad del vehículo en el eje Z
wheelbase	Numérico	De 0 a 1	Representa la distancia entre eje delantero y trasero
y	Numérico	Positivo y negativo	Posición Y en coordenada cartesianas
x	Numérico	Positivo y negativo	Posición X en coordenada cartesianas
z	Numérico	Positivo y negativo	Posición Z en coordenada cartesianas
yaw	Numérico	Positivo y negativo	Valor que indica la inclinación del vehículo
steer	Numérico	De -∞ a ∞	Ángulo del volante

**Los ángulos roll, pitch y yaw representan las tres rotaciones sobre los ejes X, Y y Z, respectivamente.*

[4]Simulador Carla: es un simulador desarrollado en Python con el fin de aprovechar el desarrollo, la formación y la validación de sistemas de conducción autónoma. La página web que hace de este simulador es carla.org.

3.5.3.1 Estructura base de datos para simulador Carla

Se crea una estructura de datos para el simulador en la cual se almacena otra tabla del modelo:

Listado 3.16: Modelo del visualizador de campos para simulador Carla.

```
class conjunto_dato_carla(models.Model):
    id = models.AutoField(primary_key=True, unique=True, auto_created=True, verbose_name="
        Identificador")
    Tiempo = models.DateTimeField(verbose_name="Momento captura", auto_now_add=True)
    Viaje = models.IntegerField(verbose_name="Identificador de trayecto", validators=[
        validar_viaje], default=1)
    x = models.FloatField(verbose_name="Posicion X", default=0)
    y = models.FloatField(verbose_name="Posicion Y", default=0)
    z = models.FloatField(verbose_name="Posicion Altura", default=0)
    pitch = models.FloatField(verbose_name="Angulo de Pitch", validators=[validar_angulo],
        default=0)
    roll = models.FloatField(verbose_name="Angulo de Roll", validators=[validar_angulo],
        default=0)
    yaw = models.FloatField(verbose_name="Angulo de Yaw", validators=[validar_angulo], default
        =0)
    steer = models.FloatField(verbose_name="Angulo de Giro", validators=[validar_angulo],
        default=0)
    lon_speed = models.FloatField(verbose_name="Velocidad", validators=[validar_velocidad],
        default=0)
    x_acc = models.FloatField(verbose_name="Aceleracion", validators=[validar_aceleracion],
        default=0)
    throttle = models.FloatField(verbose_name="Acelerador", validators=[validar_pedal],
        default=0)
    brake = models.FloatField(verbose_name="Freno", validators=[validar_pedal],default=0)
    yaw_ref= models.FloatField(verbose_name="Cabeceo", validators=[validar_angulo], default=0)
    z_speed= models.FloatField(verbose_name="Velocidad en Z", default=0)
    wheelbase= models.FloatField(verbose_name="Distancia Ejes", validators=[validar_ejes],
        default=1)
```

3.5.4 Funciones para representación de datos

Una parte fundamental de todos estos datos es ser capaz de procesarlos en viajes o trayectos y poder analizarlos a posterior. Para ello se empleó un formato de extracción para Excel, JSON o CSV de tal manera que permita ser analizado por otros procesadores de datos. Para ello se generan además filtros por trayecto, por fecha y sobre qué sistema se desea exportarlo.

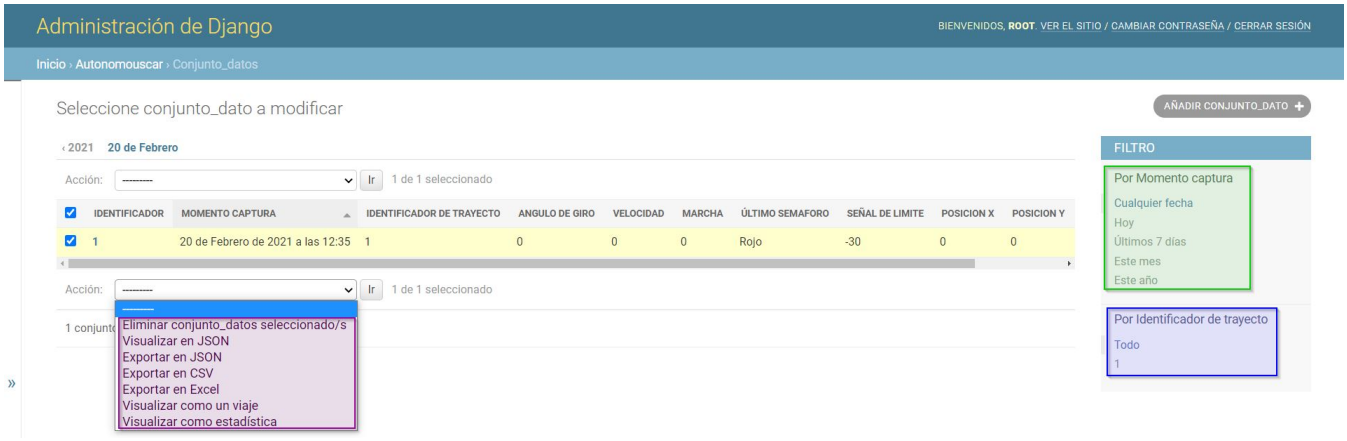

Figura 3.10: Conjunto de datos cargados, con filtros y formas de tratamiento

- Filtro por fechas: Selecciona los datos por el momento de captura.
- Filtro por trayecto: Selecciona los datos por el numero del indicador trayecto.
- Acciones para manejo de datos: para el tratamiento de cada una de las líneas de datos como un grupo.
  - Eliminación conjunto_datos seleccionado/s: Por defecto
  - Visualización JSON: Genera el texto en pantalla de un fichero JSON
  - Exportar JSON: te permite descargar la información como un fichero JSON
  - Exportar CSV: te permite descargar la información como un fichero CSV
  - Exportar Excel: te permite descargar la información como un fichero Excel, si excede el rango máximo de Excel crea diferentes paginas hasta completa toda la información
  - Visualización como un viaje: te permite ver los datos seleccionados como un viaje.
  - Visualización como estadísticas: te permite ver los datos en gráficas estadísticas.

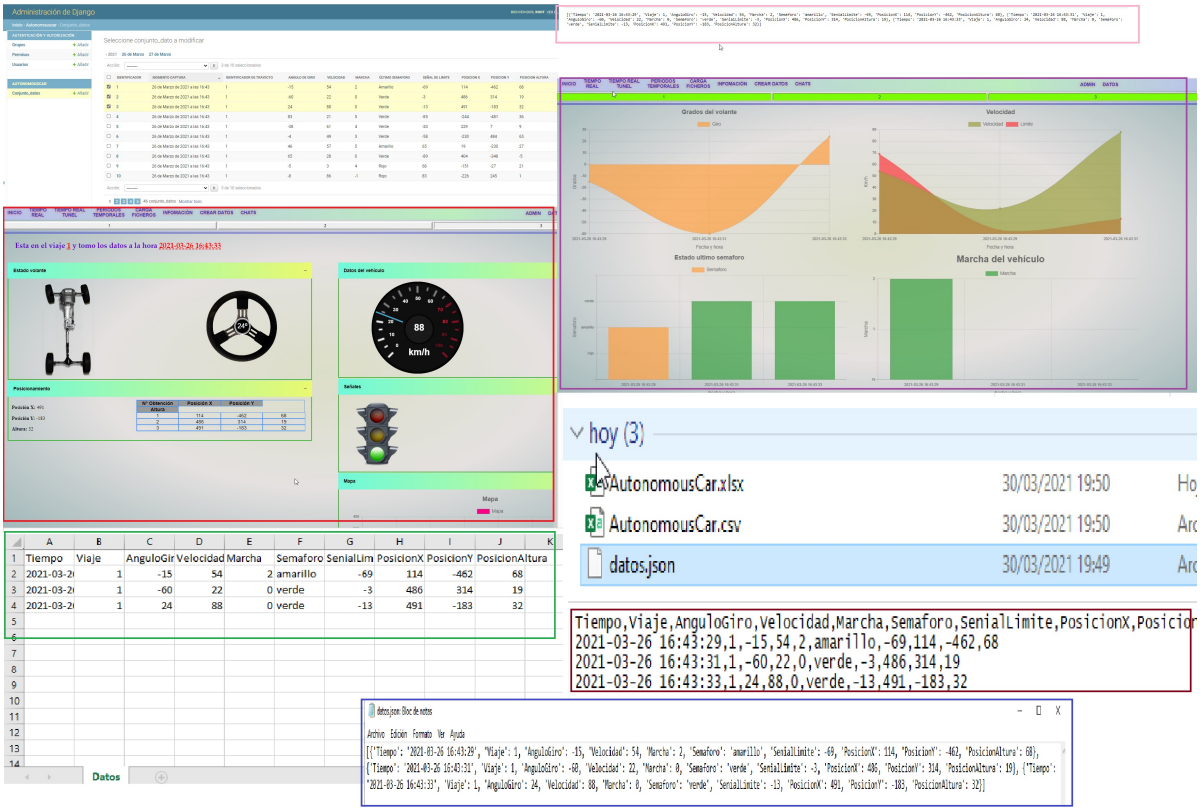

Figura 3.11: Exportaciones disponibles desde el panel de Administración

3.6 Desarrollo de viajes

Es algo muy común querer recrear una y otra vez un viaje o trayecto en el simulador para poder ver lo ocurrido en el mismo, y así analizar las diferentes formas de actuar que tuvo el vehículo en cada situación. Debido a esto se realiza un filtrado de los viajes con los que repetir la simulación.

Figura 3.12: Listado de viajes disponibles

Para la detección entre un viaje y otro se usa una variable de configuración *margen* que encontrada al comienzo del fichero models.py. Esta variable tiene un valor prefijado de 30 minutos:

Listado 3.17: Valor de margen al que se fija el tiempo entre trayectos.

```
margen=timedelta(minutes=30)
```

Si la entrada de datos entre un viaje y otro supera esta marca de 30 minutos, automáticamente cambiará de viaje con el siguiente, este control lo realiza el automatismo mediante una entrada asíncrona necesaria para el canal Redis.

Listado 3.18: Función antes de insertar en la base de datos, calcula el número de viaje.

```
from channels.db import database_sync_to_async
from datetime import datetime,timezone,timedelta
from .models import conjunto_dato,conjunto_dato_carla

@database_sync_to_async
def valor_viaje(coche,ultima_fecha,Viaje):
    try:
        if(ultima_fecha.replace(tzinfo=timezone.utc))<(datetime.now()-margen).replace(tzinfo=
            timezone.utc):
            if(coche == 1 or coche == "1"):
                r=conjunto_dato.objects.latest("id")
            else:
                r=conjunto_dato_carla.objects.latest("id")
            if (r.Tiempo.replace(tzinfo=timezone.utc))>(datetime.now()-margen).replace(tzinfo=
                timezone.utc):
                Viaje = r.Viaje
            else:
                Viaje = int(r.Viaje)+1
    except:
        Viaje=1
    finally:
        return Viaje
```

3.6.1 Mostrar un viaje

Al mostrar un viaje lo más interesante es poder interactuar con un periodo de tiempo concreto dentro del trayecto y la también variar la velocidad de la simulación. Por ello no solo interesa poder visualizar los datos de en ese momento actual, si no poder seleccionar el instante de tiempo deseado.

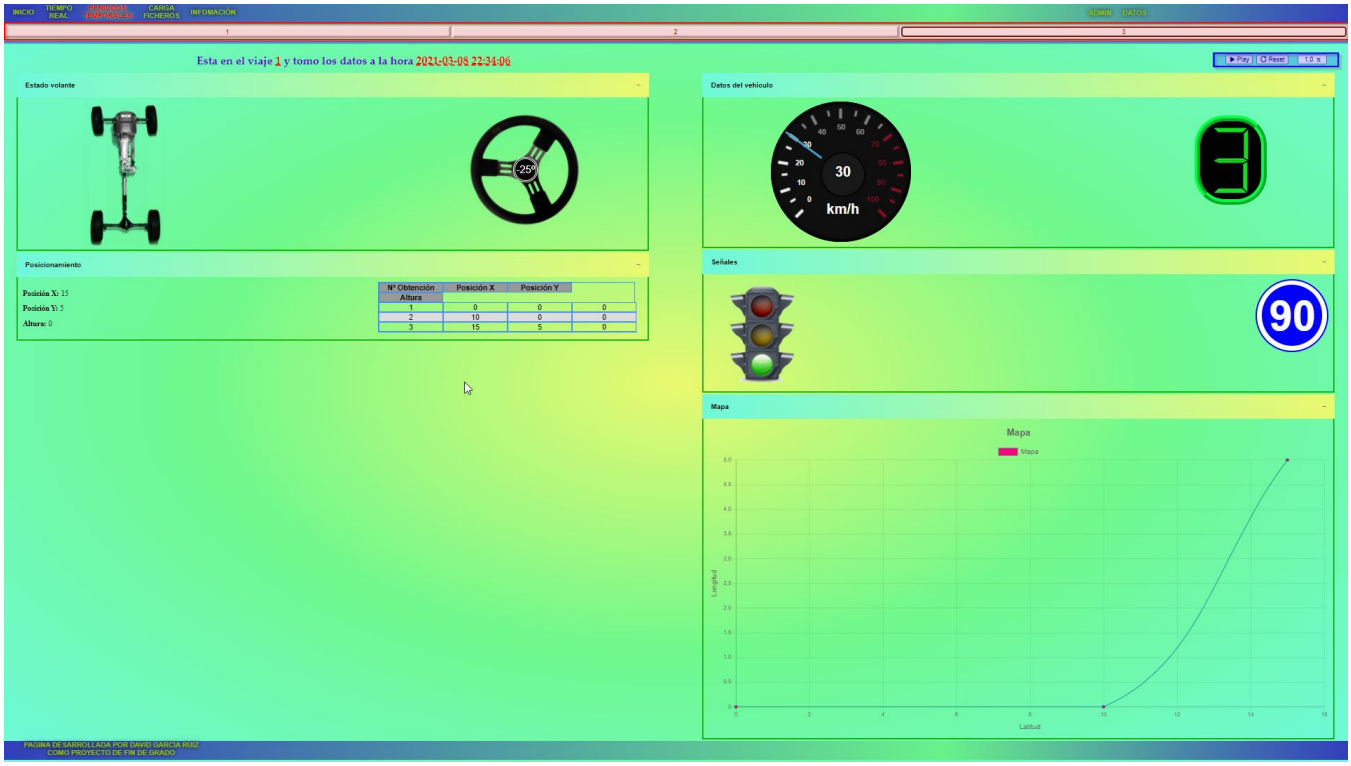

Figura 3.13: Viaje 1 con tres capturas de información

- Items del trayecto: se puede hacer clic para seleccionar el momento concreto del trayecto o para que simule desde ese punto si le damos al play.
- Velocidad simulación: se puede modificar la velocidad de simulación entre 0,5 y 10 veces comparado con la velocidad normal. También permite:
  - Comenzar la simulación desde el principio.
  - Pausar en un instante.
  - Reiniciar la simulación.
  - Detener la simulación.

Este menú solo es accesible desde el entorno de viajes del sistema y de viajes cargados desde ficheros.

3.6.1.1 Viaje creado desde la zona de administración

Desde la zona de administración 3.5.4, se puede crear un viaje a partir de los datos seleccionados tal y como se indica en la opción *Visualización como un viaje*. Se mostrará de manera similar al apartado anterior, con los datos de seleccionados. Una de las ventajas de utilizar este modo es que puedes visualizar solo un trozo del viaje o la combinación de varios viajes.

3.6.2 Carga de datos desde ficheros

Desde las acciones de descarga de datos podemos seleccionar un viaje desde la zona de administración de Django en la cual podemos descargar diferentes formatos: JSON, CSV o Excel. Antes de realizar la confirmación del cargado podemos seleccionar visualizar como un viaje o como gráficas estadísticas.

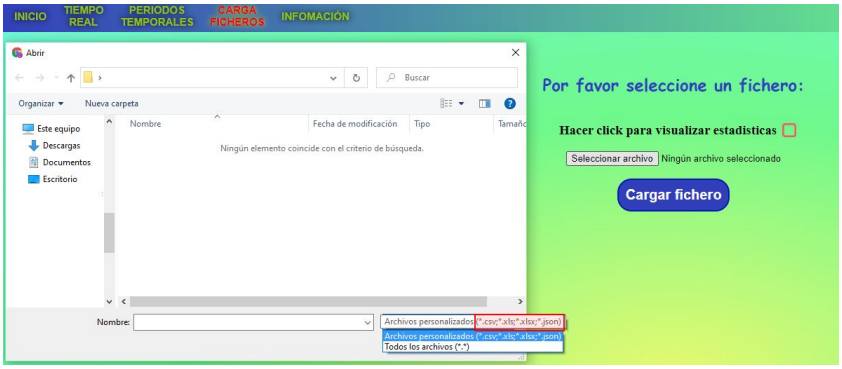

Figura 3.14: Como se pueden cargar los datos

A la hora de tratar la información la trata como un viaje sacado de la base de datos, también se pueden crear viajes a medido mediante. El proceso hace un análisis del tipo de datos y lo clasifica entre datos del sistema o datos generados por el sistema ROS del vehículo autónomo, según sea el tipo de dato y si se secciono el check de estadísticas, elegirá un tipo de visualización u otra.

3.6.2.1 Datos simulados

A partir de un fichero de los indicados en el apartado anterior, y si la estructura de datos es igual que la del modelo principal se mostrarán los datos de manera similar a los obtenidos por al seleccionar un viaje, pero podrá tener más de un valor del campo viaje (dato marcado con un recuadro rosa) ya que se convierte en otra variable del conjunto de datos.

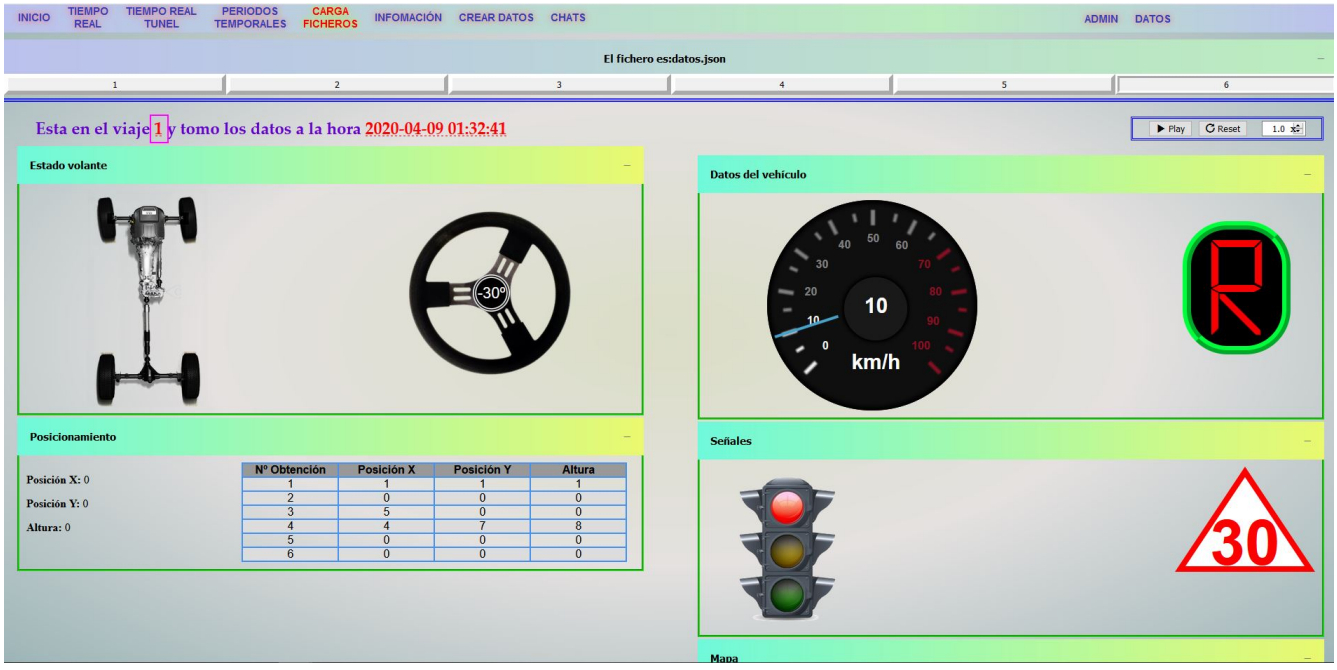

Figura 3.15: Viaje cargado a partir de fichero JSON.

3.6.2.2 Datos reales

Para poder visualizar datos cargados con información reales que proporciona por los campos que es capaz de capturar el sistema ROS encontrado en el vehículo se crea una visualización un poco diferente a la original, esta cuenta con algunos campos diferentes y algunos paneles como los de las señales, al carecer de valores, no forman parte de esta visualización.

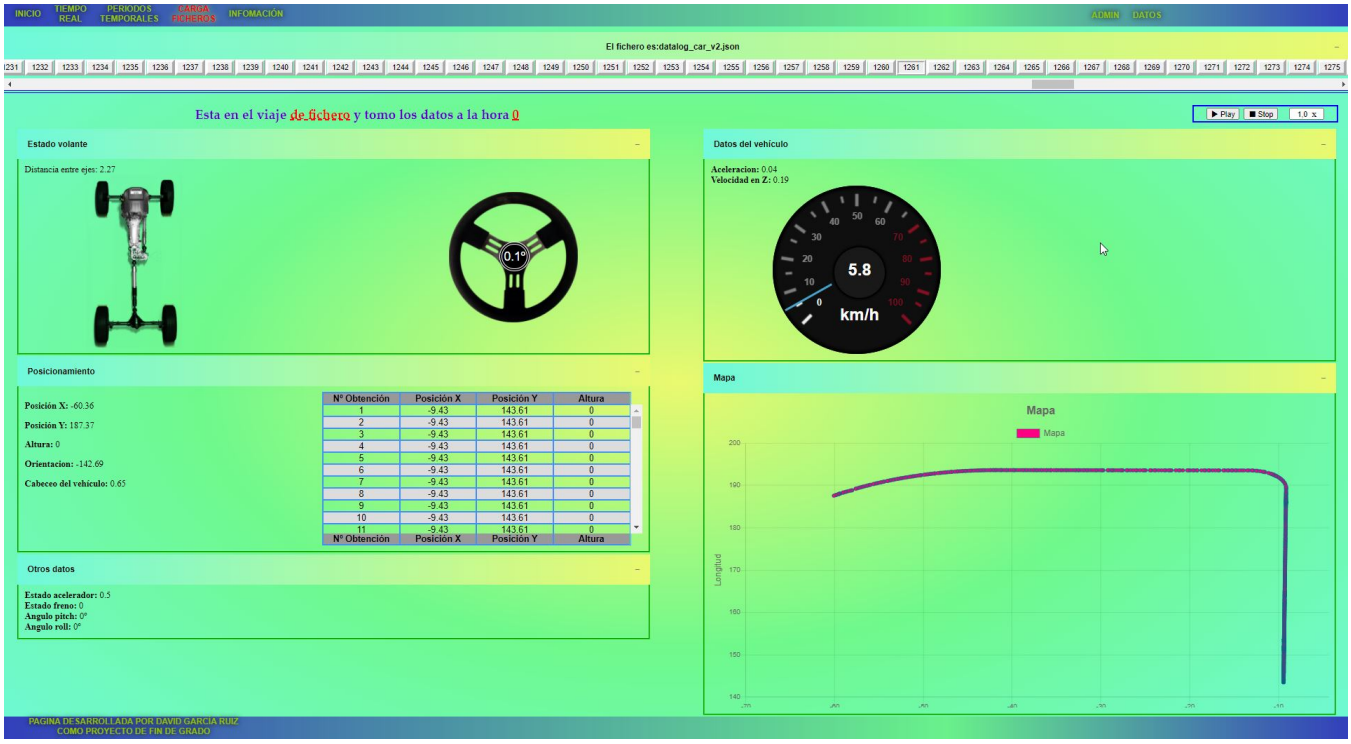

Figura 3.16: Datos del día 20-06-2020.

3.7 Desarrollo de paginas de estadísticas

Poder visualizar un contenido de datos de manera gráfica es muy útil a la hora de poder analizar visualmente lo ocurrido. Para esta representación gráfica se hizo uso de un librería de software libre llamada Chart.js[5] la en la cual se realizaron algunas modificaciones combinando las dos mejores partes de la versión 2.8 y 2.4.6. Para poder realizar representaciones, aprovechar todo el potencial de las visualizaciones en bidimensional en el formato cartesiano de un mapa fue necesario combinar librerías y de esa forma añadir todas la funcionabilidades. Se crea una sub-librería para simplificar la utilización de la librería Chart.js y tratar volumen de datos grandes con simplicidad rendimiento.

3.7.1 Implementación de la información

Partiendo de la base de envío de bloques de información para la representación de viajes, se continua usando esa misma estructura para poder tratar toda la información como un bloque y así realizar representaciones gráficas de las mismas.

[5] Chart.js[1] es una librería JavaScript que permite hacer representaciones gráficas con facilidad

3.7.1.1 Tramo seleccionable

Para el traro de ese bloque de información y con la finalidad de poder filtrar entre un rango, se puede seleccionar el principio y el fin dentro del rango.

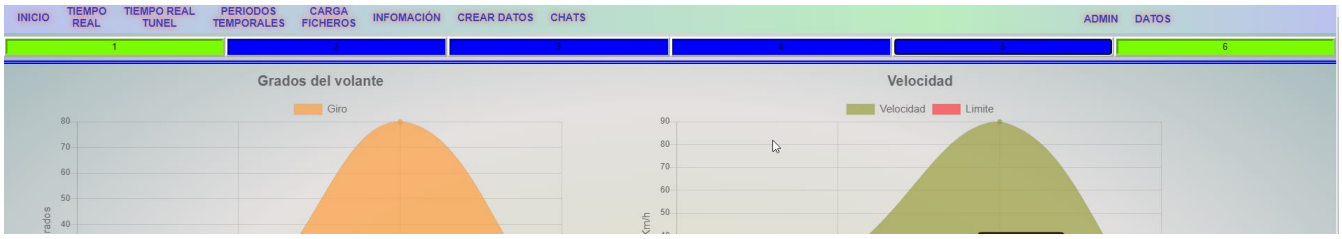

Figura 3.17: Tramo seleccionado remarcado en azul la parte mostrada.

3.7.1.2 Gráficas Lineales

En la creación de una representación lineal se hace uso de una estructura básica de un indicador de color y del color del área que abarca ese punto.

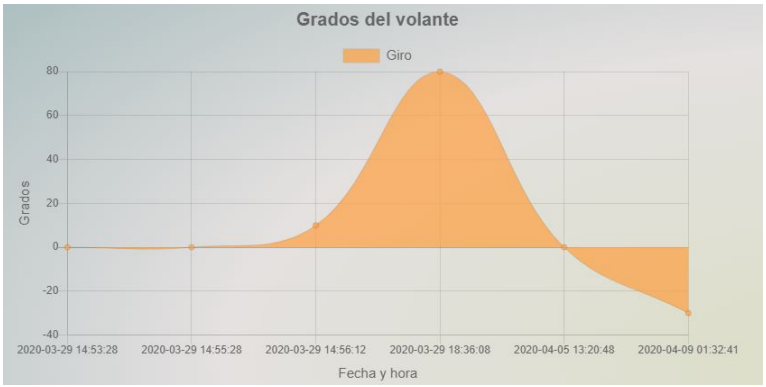

Figura 3.18: Ejemplo de gráfica lineal.

3.7.1.3 Gráficas Lineales comparativas

En la creación de una representación bilineal se hace uso de una estructura básica de un indicador de dos colores y de los colores del área que abarcan ese punto y poder comprar dos valores que estén relacionados. Ejemplo la velocidad y el límite permitido.

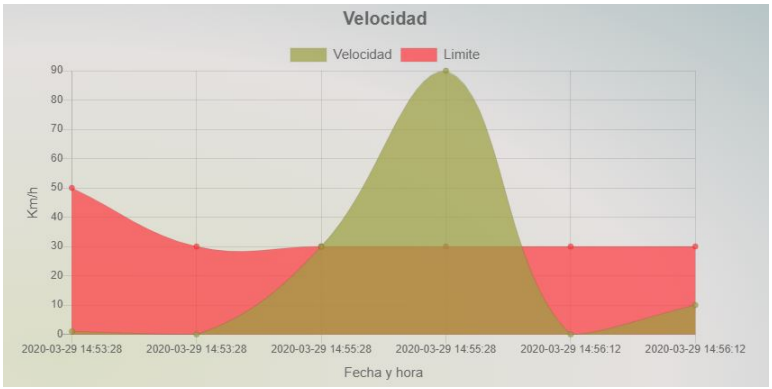

Figura 3.19: Ejemplo de gráfica bi-lineal que compara velocidad y el límite.

3.7.1.4 Gráficas Barras

En la creación de una representación de barras según los datos se hace uso de una estructura básica de un color del área de la barra. En la versión 2.4.6 no se podía inicializar las gráficas en posiciones diferentes del equivalente valor 0 en textos pero en la versión 2.8 no permitía poder indicar valores de texto en el eje Y. Se crean dos subfunciones de este tipo de gráficas:

- Gráfica semáforo: Según el color del semáforo se mostrará el color de la barra y hará referencia también a 3 niveles indicando las 3 posibles posiciones de un semáforo:
  - Rojo
  - Amarillo
  - Verde

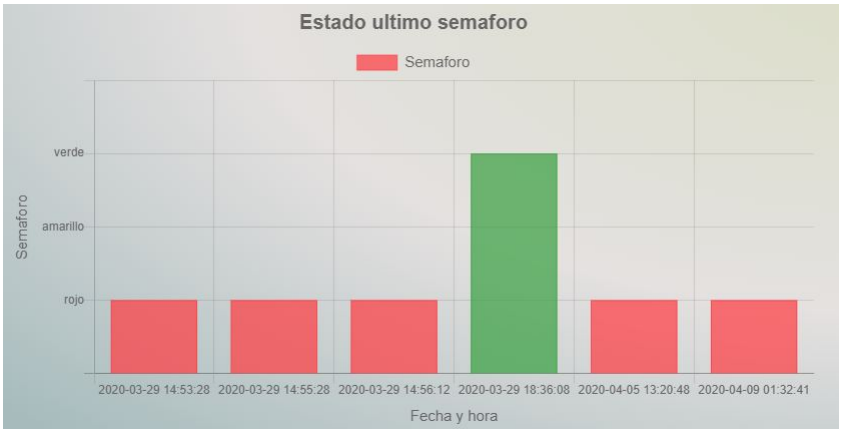

Figura 3.20: Ejemplo de representación gráfica de semáforo.

- Gráfica marcha: Según la posición de la marcha se mostrarán diferentes valores en el eje Y con los valores:

Tabla 3.3: Valores cambio de marchas.

Valores disponibles			
R	H	1	2
3	4	5	6

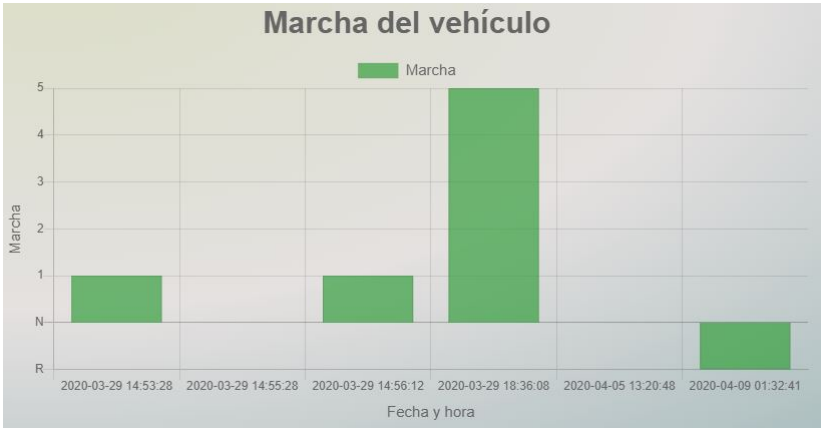

Figura 3.21: Ejemplo de representación gráfica de la posición del valor de la marcha.

3.7.1.5 Mapa cartesiano en viajes y tiempo real

Esta diagrama no poseía una composición en la versión 2.8 adecuada para realizar este tipo de representaciones cartesianas pero en la versión 2.4.6 carecía de las opciones de mostrar banner que identifica cada punto. Por lo tanto fue necesario realizar una mezcla de ambas versiones. Aprovechando este desarrollo también se hace uso de la librerías de Chart.js y se implementa una representación gráfica en el modulo "Mapa" para que se visualice la posición del vehículo en tiempo real.

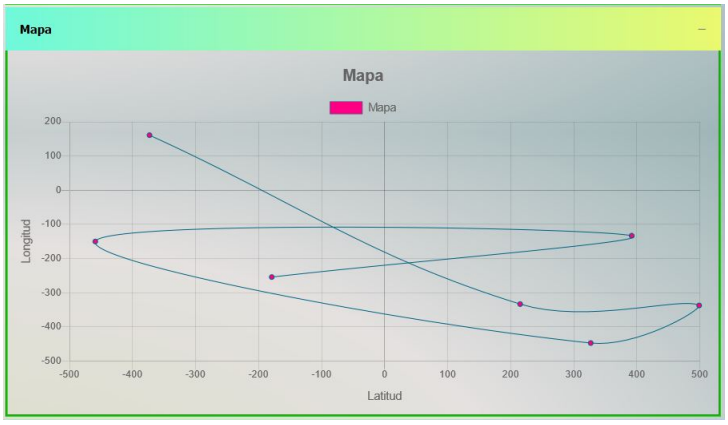

Figura 3.22: Mapa cartesiano de los ejes X e Y.

3.7.2 Datos de un viaje

Se definen una serie de tablas necesaria, y también se marca que valores no se deben representar, que valores deben evaluarse juntos y cuales es necesario que se evalúen conjuntamente (por ejemplo, las coordenadas cartesianas). El gráfico representara la información según una serie de datos para tratarlos y poder seleccionar grupos de datos. Para poder visualizar las gráficas es necesario marcar el recuadro de estadísticas en la ventana de selección de periodos temporales o carga de ficheros.

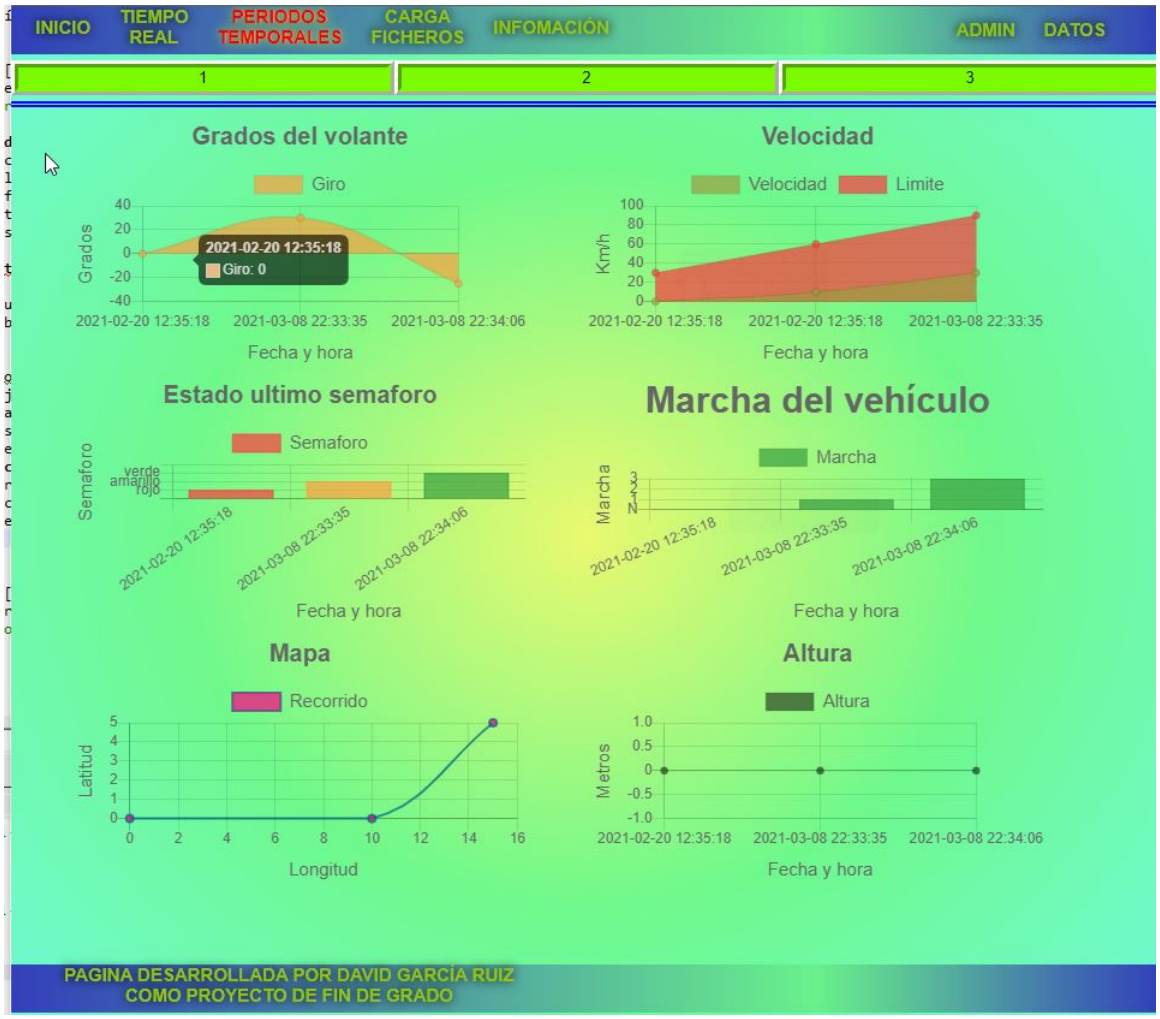

Figura 3.23: Gráficas de datos de un viaje.

3.7.2.1 Datos reales

En la forma de visualizar los datos reales existen otros campos para visualizar y dado que las tablas se generan de manera automática para las no definidas y no excepcionadas sin afectar al aspecto general.

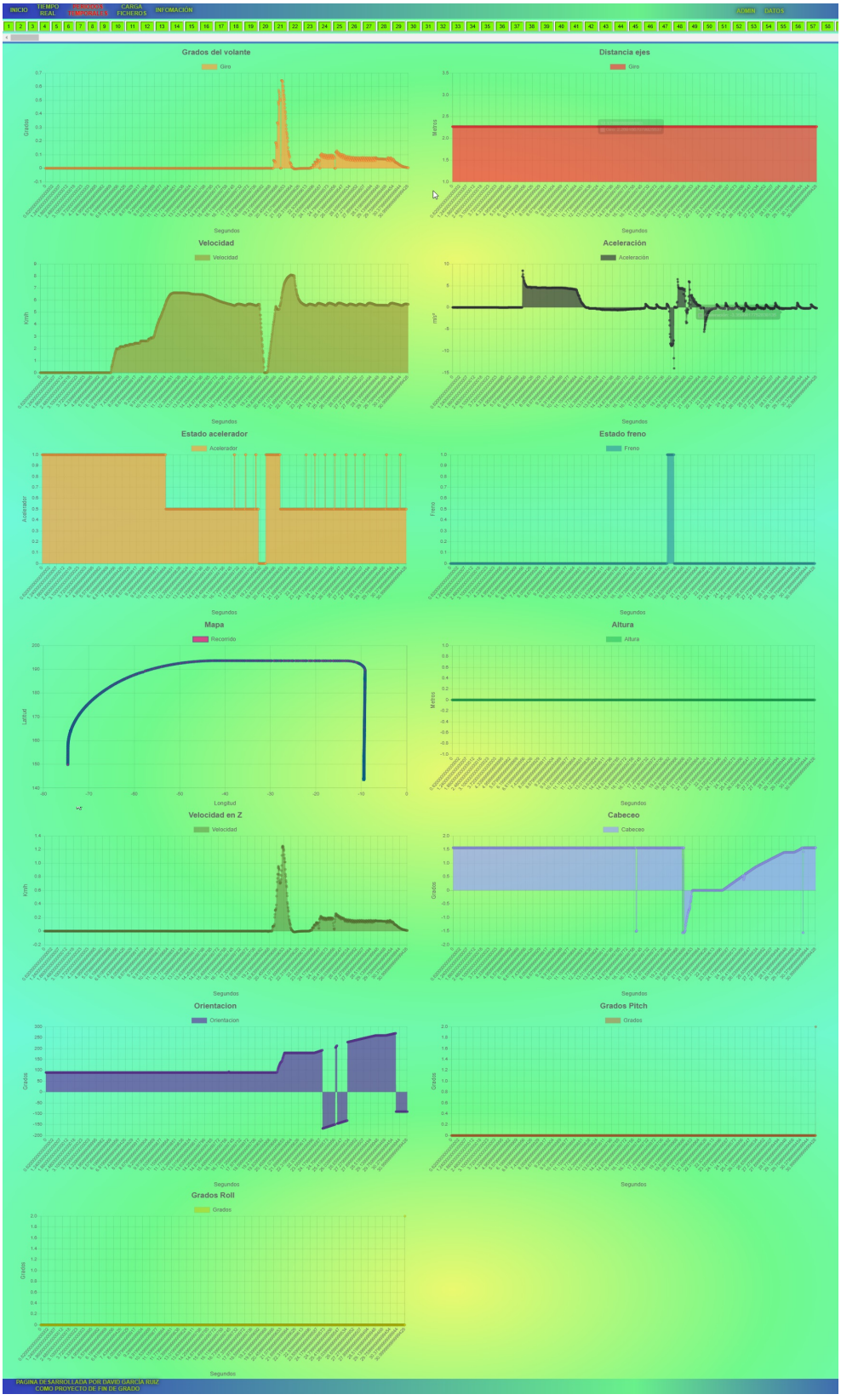

Figura 3.24: Estadísticas del día 20-06-2020.

3.8 Desarrollo de otras paginas

Toda página web suele tener una pantalla de inicio y otra de información en la cual se describe datos relevantes de la misma.

3.8.1 Pantalla inicio

En la pantalla de inicio se muestra los links principales disponibles de la web:

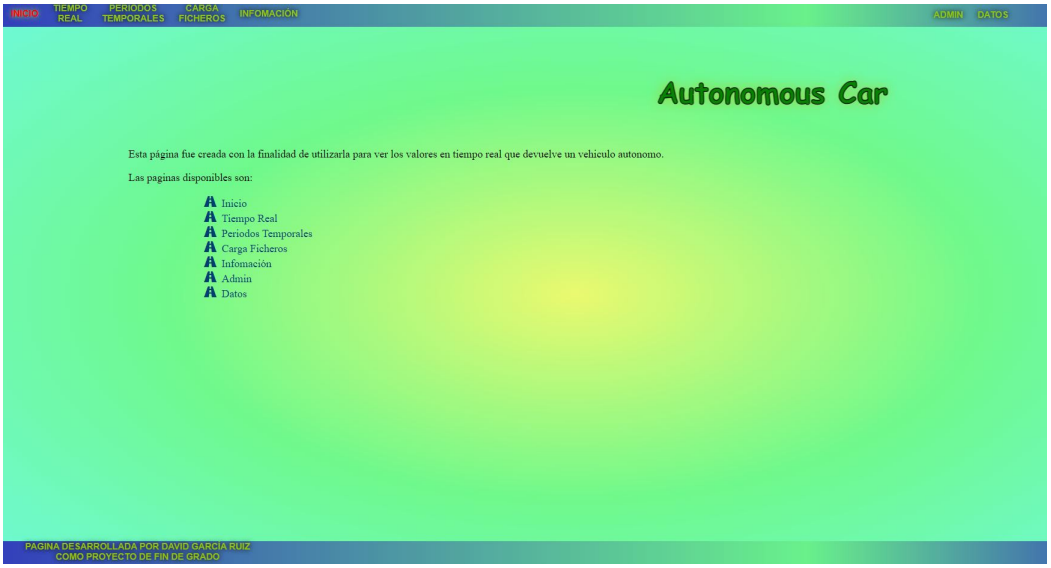

Figura 3.25: Página Inicio con links básicos.

3.8.2 Pantalla información

En esta página se explica de librerías básicas de Python necesarias para el funcionamiento de la web desarrollada sobre Django.

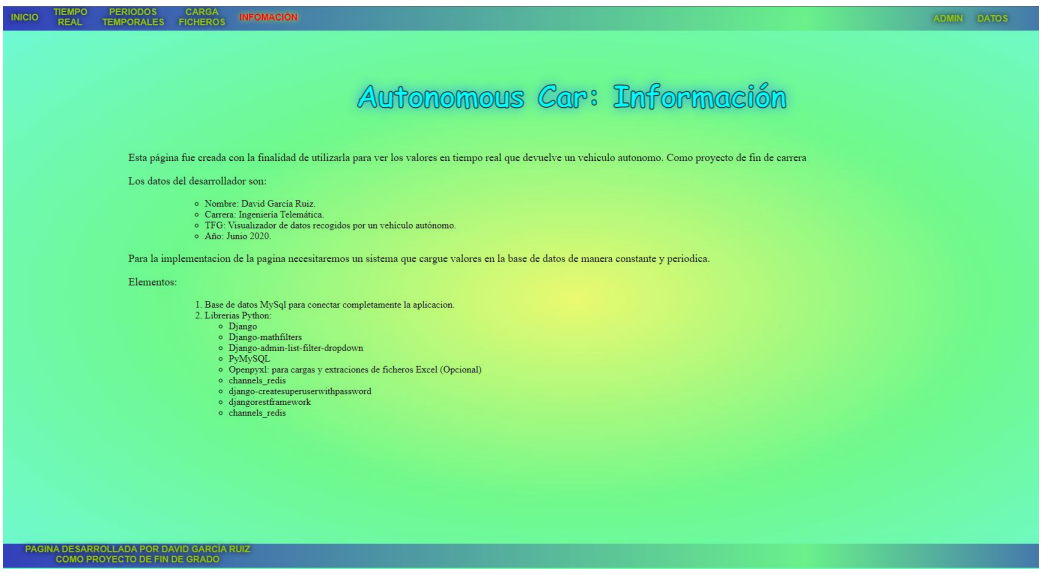

Figura 3.26: Página información de librerías mínimas necesarias para su funcionamiento.

3.9 Desarrollo de servicio rest para carga de datos

Una parte importante del proceso es disponer de un servicio desde el cual se pueda crear datos al sistema, para ello se crear un servicio Rest el cual realizara cargas de los datos sobre la plataforma sin tener que acceder directamente sobre la base de datos.

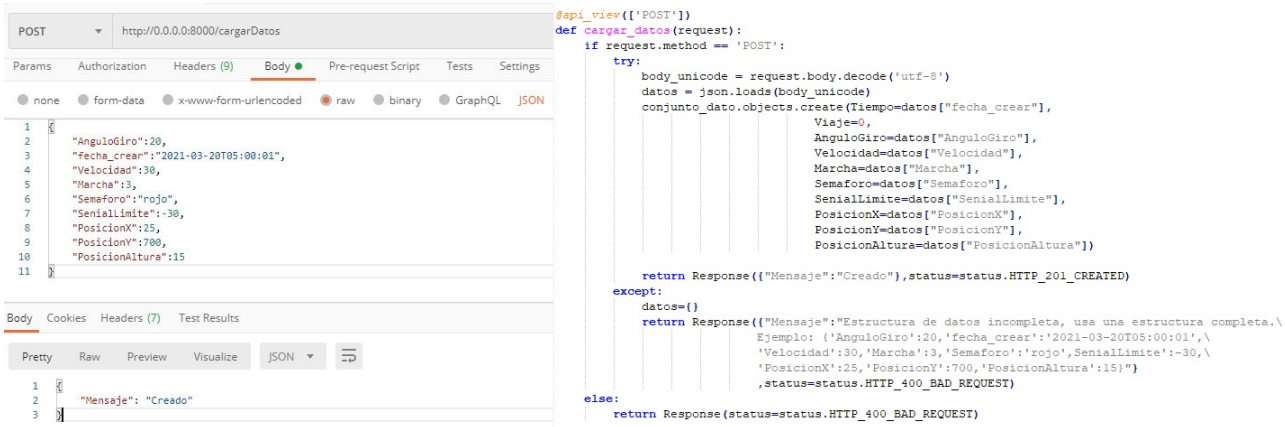

Figura 3.27: Código y ejemplo de carga de datos.

Mediante el una *Request* de tipo POST se puede almacenar datos en la base de datos en si cumple con la estructura deseada de campos necesarios para la creación y la confirmación del alta en la base de datos. Esta *Request* devuelve un código 201 indicando creado, un código 400 indicando, incluso, un ejemplo de cómo debe ser la estructura de datos o un código 404 cuando se intenta hacer una petición de otro tipo diferente a POST.

3.10 Mejora de rendimiento y comunicación del servidor web mediante canales

Como mejora del proceso básico seria la utilización de un canal para evitar saturaciones en el número de llamadas al servidor, ya que sufre tanto el servidor como el propio navegador. Además si existen multiples conexiones a la página web desde diferentes clientes, puede sufrir una saturación que cause la caída del servicio o se pueda considerar como un ataque y se bloquee la comunicación.
Para que se refleje en tiempo real la información del vehículo sin provocar esta saturación se propone usar una comunicación con el servidor mediante un canal Redis que permita mantener una conexión abierta entre el servidor y el navegador cliente.

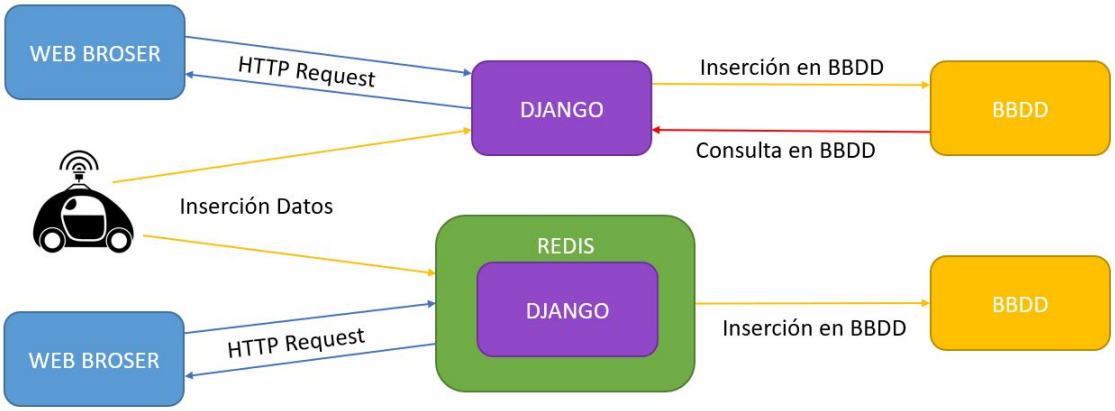

Figura 3.28: Diagrama comparativo con o sin canal de comunicación Redis.

Las comunicaciones y el número de llamadas entre los diferente sistema involucrados se ven reducidas y causan una reducción de saturación de la base de datos como indica el diagrama. A su vez al estar conectado a un canal se reduce el numero de llamadas entre el Web Browser (navegador cliente) y el servidor Django.

3.10.1 Chat

Para aprender a utilizar correctamente el funcionamiento de los canales, el desarrollo partió del desarrollo de un chat de comunicación en el cual se pudiera información entre dos navegadores clientes con acceso a la misma sala o grupo del canal.

Figura 3.29: Chat entre dos usuarios en la sala_1.

Cuando desde un navegador se envía un mensaje al servidor indicando que usuario escribe en ese servidor, este responde al resto de usuarios o navegadores clientes conectados en al servidor con el mensaje de recibido. En la imagen solo refleja 2 usuarios pero no sufre ninguna limitación número de usuarios, tampoco se comprueba el nombre de usuario. También la estructura creada de grupo de salas permite que existan infinitas posibilidades de salas diferentes en las cuales solo las que estén conectadas a la misma sala reciban mensajes de actualización. Si desde la sala *SALA-1* se manda un mensaje, este mensaje no llegará a a ningún usuario conectado en la sala *SALA-2* y tampoco viceversa.

3.10.1.1 Acceder a la sala

Para poder acceder a una sala se desarrollan dos maneras:

- Mediante URL:

Listado 3.19: Url de conexión a sala de chat.

```
http://URL_BASE/chat/[Nombre_Sala]/ Ejemplo: http://127.0.0.1:8000/chat/sala_1/
```

- Mediante la página de Chats:

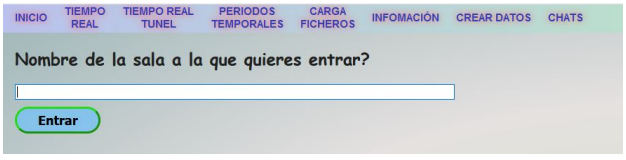

Figura 3.30: Página de acceso a la sala deseada.

3.10.2 Página tiempo real con túnel

Tras la creación del chat y tras algunas adaptaciones para enviar conjuntos de datos y con el fin de reducir el número de llamadas que recibe el servidor y saturar por el número de llamadas que recibe el mismo y la base de datos de la aplicación.
Como solución a este posible inconveniente es usar conexiones mediante canales de comunicación, en las que no es necesario estar haciendo consulta constantemente al servidor y base de datos. Además el hacer llamadas cada X tiempo no es tan real la actualización como al hacerlo por canal, porque si tengo dos actualizaciones en un segundo al hacer la petición cada segundo solo sufrirá un cambio en el visualizador y no será capaz de percatarse de porque frenaba en ese segundo, mientras que si se realiza por el otro método, se actualizará dos veces la página dando una pequeña fracción de segundo a percibir que el semáforo estaba en rojo y por eso redujo la velocidad el auto.
Otra posible solución para mitigar que se perciba ese cambios seria reducir el tiempo de muestro a la mitad (también con el dilema de cuál es el tiempo ideal de muestro de esa información para considerarla relevante y consecuente al cambio), causando el doble de consultas al servidor y causando lo definido como problema a largo del planteamiento de necesidad de este sistema. La saturación por exceso de llamadas, además de que las llamadas de consulta crecen de manera proporcional al número de conexiones y esto puede llegar a un rechazo de llamadas a la base de datos dado que esta también tiene consultas, puede causar fallos en las comunicaciones entre el vehículo y la base de datos. Con un canal la comunicación entre el vehículo y la base de datos se ve casi como un planteamiento de unicidad que solo recibe una llamada en la primera petición de apertura de la comunicación y el resto de las actualizaciones de datos los navegadores son actualizados por la información enviada por el vehículo a través del canal y esta se almacena la base de datos.

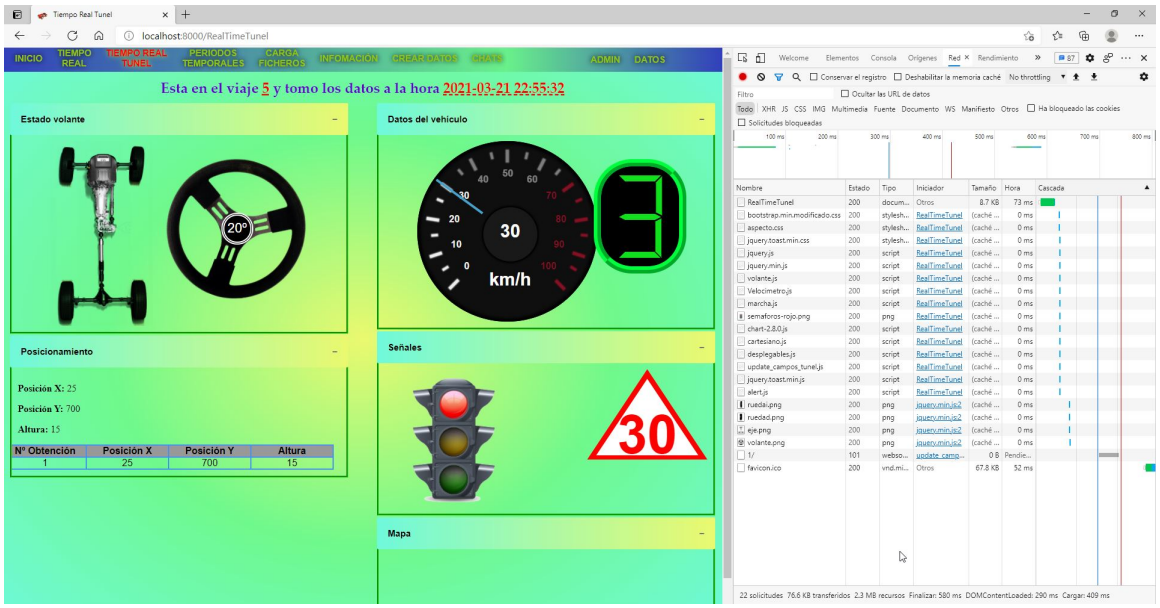

Figura 3.31: Llamada a un canal en la cual no se consume llamadas extra cada X tiempo, solo cuando reciba mensajes de actualización del servidor.

En el diagrama de red del navegador se puede observar que tras un tiempo y al no recibir ninguna petición constante por el muestreo generado por una función repetitiva que solicite datos la saturación de la red es prácticamente por parte de la aplicación, está solo se vería afectada si hubiera una actualización por parte del vehículo autónomo asociado a este canal de datos.

3.10.3 Carga desde web con Redis

Se desarrolla una página que se conecta al canal del modelo del ***coche 1*** que permite realiza carga de datos de manera que actualiza la web en tiempo real, tanto al canal Redis en la web de tiempo real túnel, como almacenarlo en la base de datos y que se vea reflejado en la petición posterior del la web de tiempo real. Mediante esta página podemos realizar simulación de un vehículo en tiempo real para simular su funcionamiento. También dispone de un botón que permite generar valores aleatorios o bien se pueden realizar ediciones de los valores para introducir el valor deseado.

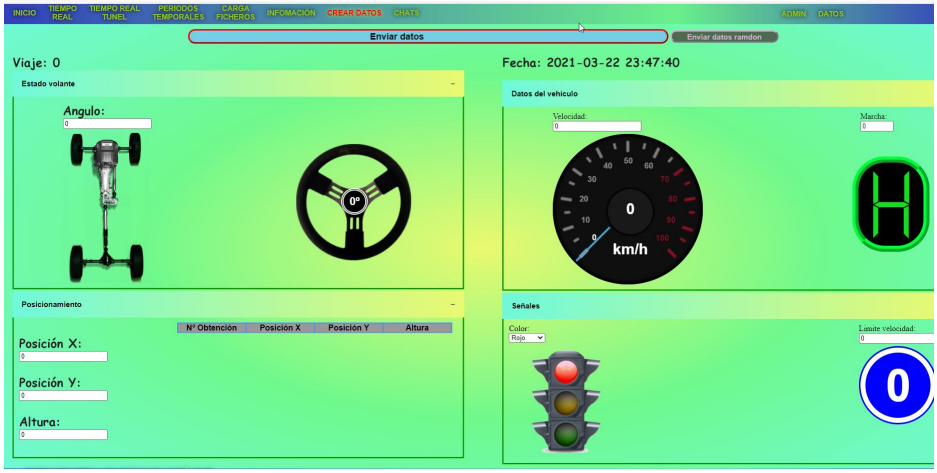

Figura 3.32: Página web para cargar datos en la web simulando el vehículo.

Los campos disponibles son:

1. Angulo: entre -90 y 90
2. Posiciones: en 3D del vehículo.
   (a) Posición X: Positivo y negativo.
   (b) Posición Y: Positivo y negativo.
   (c) Altura(Posición Z): Positivo y negativo.
3. Velocidad: Valores 0 a infinito.
4. Marchas: R a 6
5. Color del semáforo: Rojo, verde o amarillo
6. Límite de velocidad: para valores negativo prohibición limitante de velocidad y para valores positivos velocidad recomendada.

3.10.4 Carga desde Python

Se desarrolla un script de Python para conectarse al canal y poder introducir valores necesarios. Este script puede ser aplicado por ejemplo a los datos de suministrados por el simulador de Carla dado que posee un entorno en el que validad múltiples entornos, situaciones ambientales, sensores y dinámicas básicas de circulación (peatones, otros vehículos, semáforos, carriles de ambos sentidos, colisión entre vehículos y objectos, ect.). Para la utilización de este script serán necesarias las librerías datetime, json y websocket.

Listado 3.20: Código para cargar datos desde un script en Python.

```
1
2  from websocket import create_connection
3  import json
4  from datetime import datetime,timezone
5  import time
6
7  url="127.0.0.1:8000"
8
9  ws_url="ws://"+url+"/ws/coche/2/"
10
11 #Para establecer la llamada
12 ws = create_connection(ws_url)
13
14
15 def enviar_datos(ws_conect,datos):
16     try:
17         ws_conect.send(json.dumps({"message":datos}))
18         print("Datos enviados")
19     except:
20         try:
21             #Reconexion WS
22             ws_conect=create_connection(ws_url)
23             enviar_datos(ws_conect,datos)
24         except:
25             print("No se pudo almacenar los datos:")
26             print(datos)
27     return ws_conect
28
29
30 #estructura de datos
31
32 ws=enviar_datos(ws,conjunto_datos_carla)
33
34 #Al finalizar
35 ws.close()
```

La estructura de datos que tendría para poder utilizar el código de la parte superior para el coche 1 (datos sistema principal) utilizando el script sería:

Listado 3.21: Ejemplo de cargar datos al visualizador.

```
# La url a usar es: ws_url="ws://127.0.0.1:8000/ws/coche/1/"
conjunto_datos={
    "AnguloGiro":0,
    "Velocidad":0,
    "Marcha":0,
    "Semaforo":"rojo",
    "SenialLimite":0,
    "PosicionX":0,
    "PosicionY":0,
    "PosicionAltura":0,
    "fecha_crear":datetime.now(timezone.utc).isoformat(timespec='seconds')
    }

ws=enviar_datos(ws,conjunto_datos)
```

La estructura de datos que tendría para poder utilizar el código de la parte superior para el coche 2 (datos registrados en el simulador Carla) utilizando el script sería:

Listado 3.22: Ejemplo de carga con datos del simulador Carla.

```
# La url a usar es: ws_url="ws://127.0.0.1:8000/ws/coche/2/"
conjunto_datos={
    "Viaje":0,
    "AnguloGiro":0,
    "Velocidad":0,
    "Marcha":0,
    "Semaforo":"rojo",
    "SenialLimite":0,
    "PosicionX":0,
    "PosicionY":0,
    "PosicionAltura":0,
    "fecha_crear":datetime.now(timezone.utc).isoformat(timespec='seconds')
    }

ws=enviar_datos(ws,conjunto_datos)
```

3.11 Conclusiones

Los desarrollos de esta pagina web se han realizado sobre un cambio constante, realizando muchas pruebas y realizando adaptaciones según se iban añadiendo componentes o funcionabilidades al sistema.
La incorporación de un canal Redis, reduce el número de llamadas de manera notable, es una mejora importante en todo este desarrollo, parece poco importante, pero también la propia memoria del navegador cliente sufre menos con esa reducción de llamadas al servidor.
La creación de la pantalla de carga de datos, el servicio Rest para conectar el automatismo, así como los sistemas de mensajería y actualización en tiempo real del vehículo mediante *Web Service* facilitan el poder realizar pruebas del sistema completo.

Capítulo 4

Resultados

Los grandes resultados requieren grandes ambiciones

Heráclito

4.1 Introducción

En este capítulo se introducirán los resultados más relevantes del trabajo.

La estructura del capítulo es:

- Entorno Experimental
  - Base de datos utilizadas
  - Resultados Web
  - Resultado web con el canal Redis
- Autocomposición en contenedor Docker
- Resultado experimentales
- Conclusiones

4.2 Entorno experimental

El entorno puede realizarse sobre cualquier maquina en la que esté disponible un sistema Linux o una virtualización del mismo. También se puede realizar sobre entornos Windows controlados, aunque suelen tener mayor cantidad de problema debido a que la adaptación de las librerías de Python suelen estar precompiladas para entornos Linux (surgen problemas con librerías MySQL y PyMySQL). Para la utilización de un entorno es todos los puntos descritos en el Aparado 1.3. También es necesario crear la base de datos y relacionarla con el modelo del proyecto.

4.2.1 Bases de datos utilizadas

Para poder vincular la base de datos con el proyecto es necesario ejecutar una serie de comandos citados en el Apartado 3.5.2. Con la ejecución de esos comandos se crean las tablas básicas de Django y la específica del proyecto:

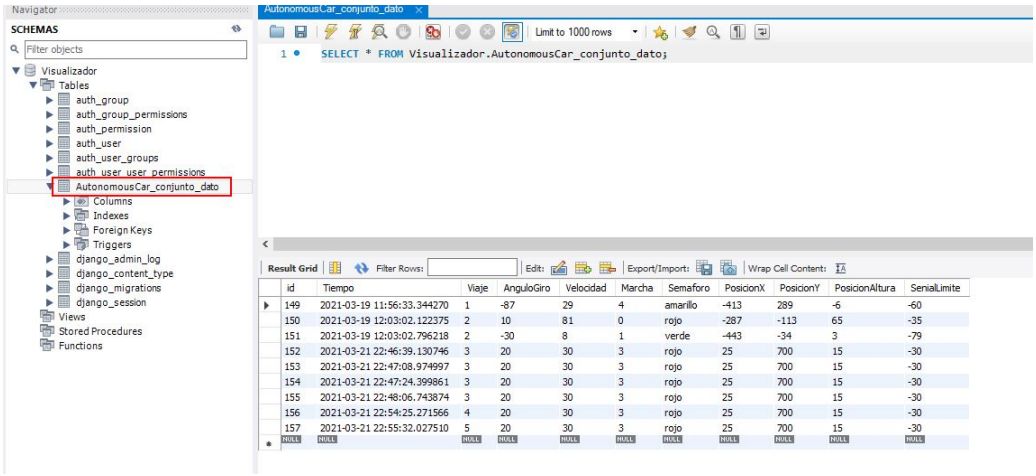

Figura 4.1: Esquema de la base de datos en MySQL y algunos datos en la tabla del proyecto.

4.2.2 Resultados Web

Para inicializar el proyecto podemos hacer uso del motor de Django utilizando el comando:

- python TFG/manage.py runserver 0.0.0.0:8000

Gracias a este comando el proyecto quedara iniciado y podremos ejecutar las diferentes ventanas del proyecto.

Figura 4.2: Todas las páginas del automatismo.

4.2.3 Resultado web con el canal Redis

Gracias al canal Redis podemos comparar los resultados en apenas 3 minutos en los cuales se puede ver como el caso de tiempo real realiza 180 llamadas aproximadamente frente a las 20 realizadas por el tiempo real túnel el cual solo mostrar más comunicaciones de red cuando sufra una actualización la por parte del vehículo.

Figura 4.3: Comparativa de llamadas en 3 minutos, aproximadamente, sin actualizaciones del auto.

4.3 Autocomposición en contenedor Docker

Tras enfrentarme a la dificultad de la instalación de algunas librerías de Python en un sistema Windows, decidí utilizar un sistema virtualizado que fuera compatible tanto en Windows como en Linux y fuera posible de ejecutar en ambos entornos. Para poder crear el contenedor es necesario:

- Fichero Dockerfile

Listado 4.1: Comandos del Dockerfile.

```
FROM python:3.8-alpine

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1
RUN apk add --update --no-cache mariadb-dev jpeg-dev
RUN apk add --update --no-cache --virtual .tmp-build-deps \
        gcc libc-dev linux-headers python3-dev musl-dev zlib zlib-dev
RUN apk add openssl
RUN apk add --update alpine-sdk && apk add libffi-dev openssl-dev
RUN apk add gcc musl-dev python3-dev libffi-dev openssl-dev cargo
env ARCHFLAGS="-arch x86_64" LDFLAGS="-L/usr/local/opt/openssl/lib" CFLAGS="-I/usr/local/opt/
    openssl/include"
WORKDIR /TFG
COPY * /TFG/
COPY requirements.txt /TFG/
RUN /usr/local/bin/python -m pip install --upgrade pip
RUN pip install cryptography
RUN pip install -r requirements.txt

USER root
RUN chmod +x ./manage.py
```

- Fichero docker-compose.yml

Listado 4.2: Fichero docker-compose.yml para la creación de Redis, MySQL y Django.

```
version: "3.8"

services:
  redis:
    image: "redis:alpine"
    command: redis-server

    ports:
     - 6379:6379
  db:
    container_name: mysql_pre
    image: mysql:8
    environment:
       - MYSQL_ROOT_PASSWORD=tfgadmin
       - MYSQL_DATABASE=Visualizador
       - MYSQL_USER=tfg
       - MYSQL_PASSWORD=password
       - MYSQL_HOST=''
    ports:
      - 33060:3306

  web:
    container_name: TFG_1
    build: .
    command: python TFG/manage.py runserver 0.0.0.0:8000
    ports:
      - "8000:8000"
    volumes:
      - .:/TFG
      - /tmp/TFG/mysqld:/run/mysqld
    depends_on:
      - db
      - redis
```

- Y después ejecutando algunos de estos comandos, para preparar, crear el contenedor Docker, iniciarlo, pararlo, resetear un módulo, preparar el modelo, crear el modelo en el MySQL y la creación del superusuario de administración de Django:

Listado 4.3: Comandos para manejo del contenedor Docker y preparación del MySQL.

```
docker-compose build
docker-compose create
docker-compose start
docker-compose stop
docker restart TFG_1
docker-compose run web python TFG/manage.py makemigrations
docker-compose run web python TFG/manage.py migrate
docker-compose run web python TFG/manage.py createsuperuserwithpassword --username root --
    password root --email root@root.com --preserve
```

4.4 Conclusiones

Lo bueno que tiene el proyecto y el desarrollo sobre esta plataforma es que puedes ir visualizando la mayoría de los resultados de cada una de las partes sin que tengas que realizar todo el conjunto, sin necesidad de esperas por largas compilaciones, de forma que es más fácil y rápido para corregir los pequeños errores que surgían. Por ello los resultados se ven poco a poco y fomentan la energía de un mismo a continuar.

Capítulo 5

Conclusiones y líneas futuras

Rem tene, verba sequentur (Si dominas el tema, las palabras vendrán solas).

Catón el Viejo

En este apartado se resumen las conclusiones obtenidas y se proponen futuras líneas de investigación que se deriven del trabajo.

La estructura del capítulo es:

- Conclusiones generales
- Líneas futuras

5.1 Conclusiones Generales

El proyecto me ha ayudado a conocer gran parte del potencial del desarrollo modulable de una página web, el desarrollo sobre el Framework Django de Python y todos uso de mis conocimientos adquiridos a lo largo de mi carrera y vida profesional. Como son las bases de datos MySQL, bases de datos Redis, Sockets o canales y ampliación de conocimientos en la utilización de librerías Python utilizadas en este desarrollo.

5.2 Líneas futuras

Las líneas futuras del mejoras de este desarrollo tienen diferentes enfoques,

- Ampliación de datos a registrar.
- Mejoras visuales.
- Amplificación a mayor volumen de vehículos
- Implementar seguridad
- Ampliación a otros sectores

- Mejoras de rendimiento con sistemas mas productivos.
- Mejoras de sistema que manejen volúmenes mayores de datos.

5.2.1 Adaptación de la base de datos

Este proceso se adaptó para los datos recibidos por el simulador Carla, en función de algunas de las variables que puedes reflejar de ese simulador. Aun así se desarrolló los esquemas de pintado de los datos recibidos por un vehículo autónomo sería necesario adaptar el modelo de base de datos a esos campos que devuelva el sistema ROS.

5.2.1.1 Campos posibles a incorporar

Existen campos que no son reflejados actualmente como el volumen de la gasolina o nivel de carga de las baterías, el temperatura del aceite, nivel de aceite, presión de las ruedas, kilómetros del vehículo, etc.

5.2.1.2 Otros campos posibles a incorporar

Existen otros campos que pueden almacenarse en otra base de datos relacional en la cual se almacenen datos como fecha de cambio de la ruedas (que cambiaría con el evento de ir al taller, al hacer el mantenimiento), fechas de mantenimiento, cambios de piezas por roturas en las que quede reflejado cuando se realizaron estos cambios, tablas en las que se registren los usuarios que usan ese vehículo, o datos fijos de información del coche, como el color, marca, matrícula y el número máximo de ocupantes.

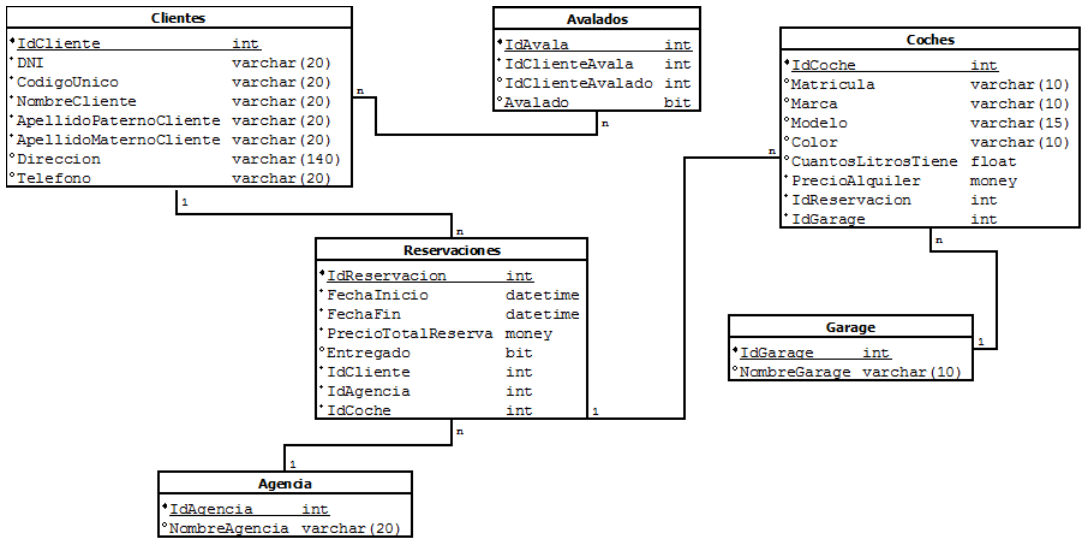

Figura 5.1: Relación base de datos de vehículos en alquiler por soloesciencia.com.

5.2.2 Mejoras visuales

Algunas de las mejoras visuales que se pueden realizar incorporando módulos nuevos que completen un conjunto de datos mayor, van enfocadas a mejoras en visualización de mapas, imágenes en tiempo real, estado del niveles de combustible y aceite, etc...

5.2.2.1 Visualización mapas

Se podría añadir algún tipo de visualizador de rutas mediante la implantación de mapas sobre los que realizar la simulación del recorrido trazado o la implementación de algún sistema existente de enrutado como Google Routes[1] o algún modulo que conecte con una cámara del vehículo.

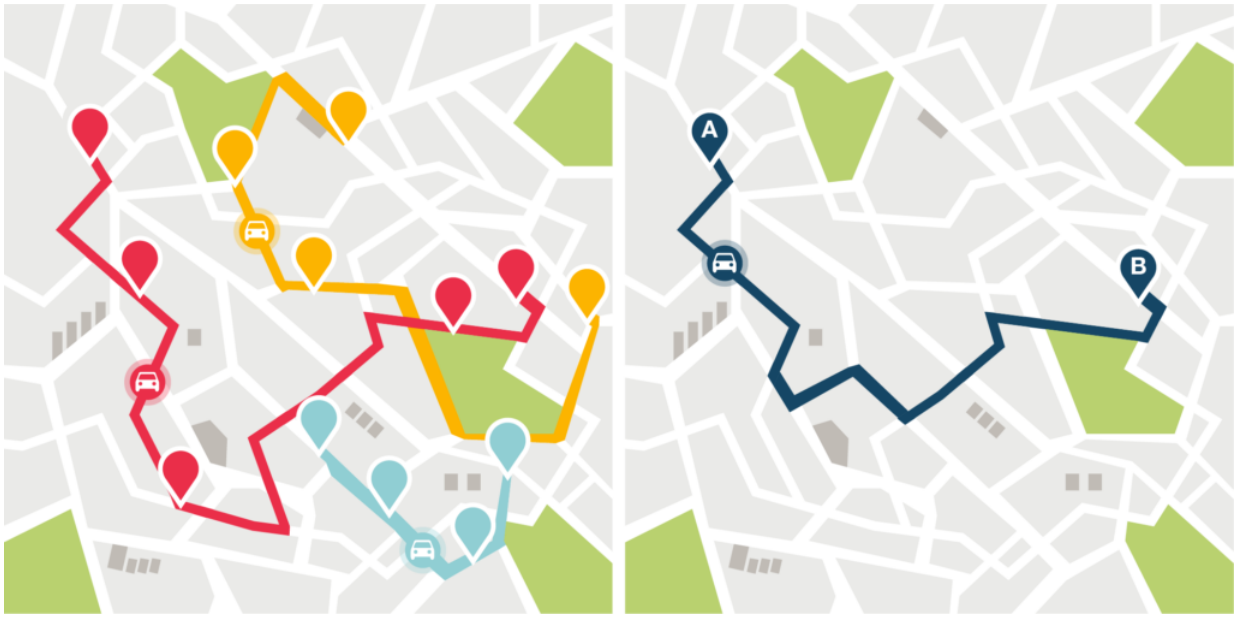

Figura 5.2: Rutas de los diferentes del coche o de diferentes coches y un trayecto.

5.2.2.2 Imágenes tiempo real

La creación de un sistema de retransmisión de datos que transmita los vídeos a sistema en tiempo real como imágenes capturadas o como un vídeo en tiempo real. Algunas referencias de cómo realizar esta retransmisión en tiempo real la encontré en Stack Overflow.

Figura 5.3: Rutas de los diferentes del coche o de diferentes coches y un trayecto.

[1] Google Routes: es una herramienta de Google que permite visualizar rutas y mapas.

5.2.2.3 Niveles de aceite y gasolina o baterías

Una información interesante es el indicador de combustible ya sea gasolina, diésel, gas, eléctrico o híbridos, y los niveles de aceite que pueden causar un fallo en el vehículo.

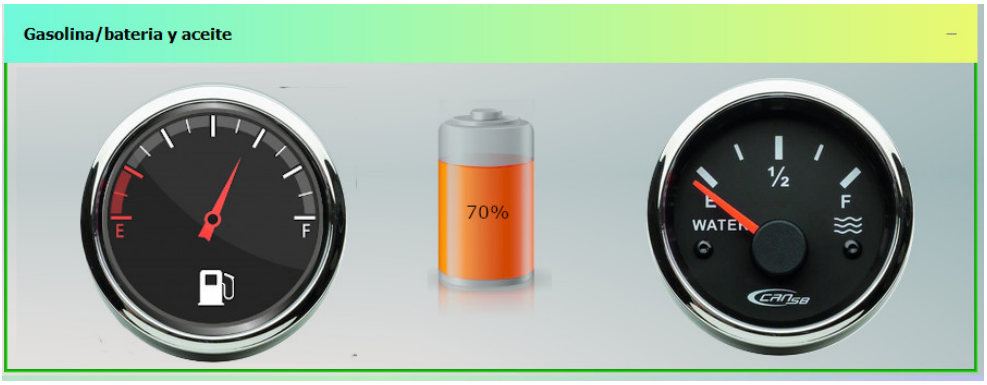

Figura 5.4: Ejemplo de módulo de gasolina, eléctrico y aceite.

5.2.3 Volumen de vehículos

Otra forma de utilizar todo el potencial que puede aportar este desarrollo es la incorporación de más de un vehículo. Lo cual podría ser relativamente sencillo haciendo pequeñas modificaciones en las consultas de petición de datos, diferenciando entre canales para cada vehículo, selectores entre vehículos y un pequeño selector de vehículos.
En caso de querer realizar esta tarea el servidor tendría que encontrarse disponible el servidor de registro para todos los vehículos y tendría que almacenar dicha información siendo enviada al servidor desde cada vehículo, esto podría conllevar perdidas del alguna información debido a la perdida de cobertura, por lo que sería conveniente que este registro también se hiciera en el propio vehículo con el fin de disponer de toda la información siempre, y después mediante los diferentes sistemas de carga que dispone la propia plataforma realizar simulaciones o poder revisar estadísticas del mismo. Si existiera en estos un sistema de gestión que permitiera la retransmisión de ese bloque de datos para analizar podría minimizarse esta perdida de datos.

5.2.4 Implementar seguridad

Para evitar posibles ataques falseando datos en los canales y con el fin de limitar el acceso a la web, se podrían gestionar roles y protocolos para comunicaciones entre el servidor y los vehículos, limitando accesos a nivel de vehículo, de niveles de gestión o utilización del vehículo.

5.2.5 Ampliación a otros sectores

La ampliación de este desarrollo a otros sectores es su enfoque a vehículos de alquiler, para conocer su estado y quienes lo usaron, como incluso la creación de un nuevo sector de transporte público para personas con vehículos autónomos, lo considero nuevo sector, ya que no existe nada parecido (Taxis sin conductor).Concretamente en España este tipo de actividad no sería bien acogida por el sector de transporte, como es sabido, existen problemas entre los antiguos taxis y los servicios VTC[2].

[2]VTC (Vehículos de Turismo con Conductor): es la denominación que reciben los vehículos con autorización en transporte con conductor, plataforma Cabify, Uber, etc

5.2.5.1 Sector agrario

También podría entrar en otro sector gestionando otro tipo de tareas, como es por ejemplo el sector agrario, en el que mediante la ampliación de la geolocalización mediante estaciones base de referencia y sistemas de comunicación que permiten dirigir vehículos agrarios (tractores, cosechadoras, empaquetadoras...) y realicen las tareas a distancia.

Figura 5.5: Sistema de guiado de un tractor de la empresa GPS New Holland AG.

5.2.6 Mejoras de rendimiento con sistemas más productivos

La implementación realizada durante este desarrollo utilizando canales Redis refleja como aprovechar este funcionamiento aunque también podría aprovecharse la propia base de datos Redis como una base de datos fija o persistente, su forma de almacenar la información es por referencias clave valor por lo tanto no esta del todo estructurada en cuanto a campos a utilizar.
Otra mejora de rendimiento seria cambiar el motor del servidor de Django por otro motor como por ejemplo Nginx[3] o similar que mejoran la velocidad de respuesta y de rendimiento que da el *manager.py* de Django, tomando como ejemplo el despliegue explicado en la web de platzi.com

5.2.7 Mejoras de sistema que manejen volúmenes mayores de datos

A la hora de manejar muchísimos vehículos y con el fin de coordinar una flota de coches dentro de un único sistema, dejando siempre la posibilidad de conexión a los datos de estos se podría generar otra estructura diferente en el modelo del proceso. Para ellos se podría realizar un gestión por eventos en lugar de por bloques de información en instantes de tiempo, aunque exista también alguna conexión directa de información para la retransmisión en tiempo real del vehículo por ejemplo.

[3]Nginx:? es un servidor we o/proxy inverso ligero de alto rendimiento.

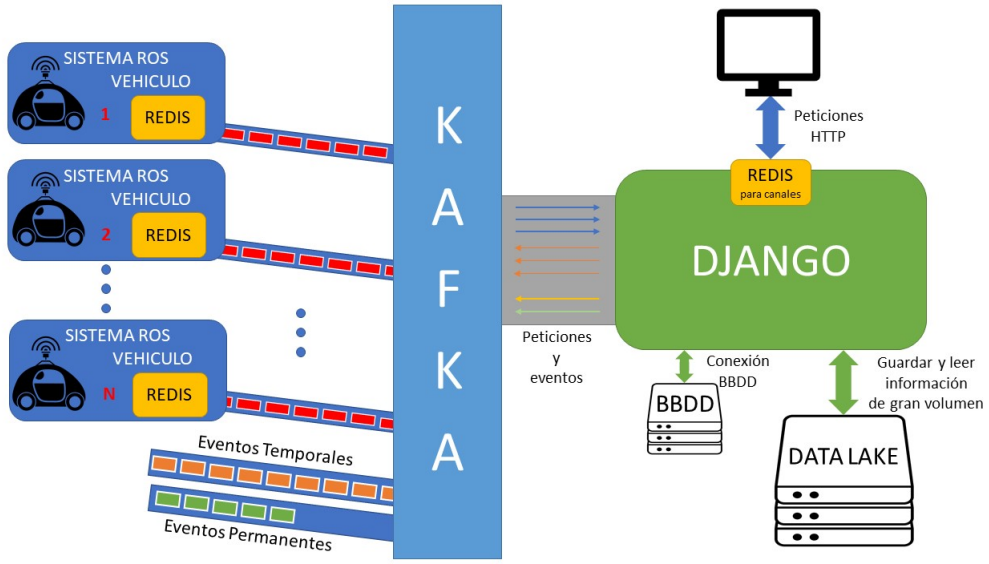

Figura 5.6: Esquema de implementación de un sistema de comunicación de los vehículos mediante eventos

- Una base de datos Redis que haga de conector de conexión para la retransmisión del vídeo y también como base de datos local en los vehículos dentro del vehículo.
- Un Redis entre el sistema Django y el consumidor de peticiones HTTP para la transmisión en tiempo real de los eventos recibidos.
- El sistema ROS de cada vehículo se encargara de almacenar la información dentro de la base de datos Redis y manejará los eventos de entrada (notificaciones del servidor con mensajes persistentes o temporales) y eventos de salida (datos del vehículo).
  - Eventos Permanente: son eventos que no tienen fecha de caducidad dentro de las colas de eventos de Kafka. Por ejemplo, un indicador de los puntos donde los vehículos deben acudir para cambiar sus baterías cuando tengan unos niveles bajos de las mismas.
  - Eventos Temporales: son eventos que se tienen una fecha de caducidad. Por ejemplo si es una petición de un alquiler de vehículos sería el evento de esa petición hasta que un vehículo la recibe y la procesa.
  - Eventos de salida: son eventos temporales hasta que los recibe el sistema Django aunque estos viajan con la información de los datos del vehículo o el resultado de alguna petición por parte del servidor.
- Django: se encarga de la gestión de los vehículos mediante el trato de los eventos enviados y recibidos por Kafka, también hará la parte de comunicación con los usuarios que consuman la información de dicho aplicativo. Los datos recibidos por los eventos generados en los vehículos, interesaría almacenarlos en una base de datos de gran volumen y por otro lado tendríamos los una conexión a una base de datos normal, donde estuviera registrada otra información más condicionada al cambio.
- BBDD: La base de datos puede ser un MySQL o MariaDB o similar que haga de conexión con datos sujetos a cambios con los que interactuará Django. Por ejemplo, la lista de usuarios que se pueden conectar, configuraciones de los eventos permanente, información sobre el estado de las baterías de los vehículos, etc.

- Data Lake: se puede considerar como una base de datos o repositorio en el que se manejan un gran volumen de datos (Cassandra[4] o HDFS[5]), por lo que se recomienda utilizar este tipo de sistemas para almacenar los datos de los vehículos. Este tipo de base de datos son muy útiles para la lectura e inserción de datos pero presenta dificultades con las modificaciones de las mismas. Con el fin de poder realizar cálculos de Machine Learning o Big Data para mejorar eficiencia de utilización de los vehículos, mejorar las condiciones de autopilotaje y el almacenamiento del gran volumen de datos que puede realizar muchos vehículos es recomendado utilizar un tipo de almacenamiento de este tipo para los datos aportados por el vehículo.

- Kafka[2]: es un sistema de gestión de colas de eventos, posee una adaptación a la escalabilidad de máquinas que generan los eventos, como a las que los procesan con gran facilidad, funciona como un único servicio con capacidad de alta disponibilidad y almacenamiento de información de manera más segura y estable que un canal Redis. Es un sistema que tienen interacción con múltiples lenguajes de programación y mediante mensajes asíncronos.

[4]Cassandra: es una base de datos NoSQL que funciona por información no estructurada con almacenamientos de datos de clave-valor, que permite almacenar un gran volumen de datos.

[5]HDFS(Hadoop Distributed File System): Es un sistema de almacenamiento muy usada en sistemas de Big Data dada su facilidad de crecimiento horizontal, que permite almacenar una gran cantidad de información y es capaz de trabajar de manera ágil al trabajar con grande bloques de datos.

Bibliografía

[1] "Página para gráficos estadisticos," https://www.chartjs.org/.

[2] "Página con la información para la implementación y utilización de kafka desde python/django," https://kafka.apache.org/documentation/# y https://pypi.org/project/kafka-python/.

[3] "Información sobre gnu/linux en wikipedia," http://es.wikipedia.org/wiki/GNU/Linux [Último acceso 1/noviembre/2013].

[4] "Página sobre procesador de texto notepath++," https://notepad-plus-plus.org/.

[5] "Página sobre interprete python," https://www.python.org/.

[6] "Página sobre depurador sypder para lenguaje python," https://www.spyder-ide.org/.

[7] "Página de stack overflow ayudo a solventar muchísimas dudas," https://stackoverflow.com/.

[8] "La página de documentación de django," https://docs.djangoproject.com/.

[9] "Página w3 schools resuelve muchas de las consultas sobre python, html, css y javascript," https://www.w3schools.com/.

[10] "La página de documentación de manejo de modelos de django," https://www.geeksforgeeks.org/.

[11] "La página de documentación de django," https://djangobook.com/.

[12] "La página de documentación y tutoriales de django," https://tutorial.djangogirls.org/.

[13] "Página para creacion de velocimetro," https://canvas-gauges.com/.

Apéndice A

Manual de usuario

A.1 Introducción

El manual de usuario de este aplicativo es bastante sencillo porque cuenta con poca interacción. La navegación por la web es bastante intuitiva, no tiene grandes complejidades de manejo. El manual de usuario se divide en:

- Visualización básica
- Visualización viajes y interacción básica
- Carga de datos y extracción de datos desde portal

A.2 Visualización básica

Las páginas de 'Inicio´, 'Tiempo Real', 'Tiempo Real Túnel', 'Información', 'Viaje cargado' y 'Viaje NºX' son páginas web solo de visualización de información.

A.3 Visualización viajes y interacción

La página de visualización de viajes permite seleccionar los datos viaje o las estadísticas del mismo. La página web de 'Chats' y 'Sala X' son páginas con campos para escribir información y hablar en salas de chat que se seleccione.

A.4 Carga de datos y extracción de datos desde portal

La página de 'Carga' permite realizar cargas de diferentes tipos de ficheros y después realizar visualizaciones de los datos o las estadísticas. Desde el portal de administración se pueden realizar las descargas de los ficheros y tratar el conjunto de datos. La página de creación de datos permite introducir diferentes valores para la simulación de los datos.

Apéndice B

Manual de instalación

B.1 Introducción

El manual de instalación describe la manera de instalar las herramientas necesarias para el funcionamiento del proceso. El manual de instalación se divide en:

- Instalación básica
- Instalación Docker
- Auto instalación con CMD

B.2 Instalación básica

Los comandos básicos serian instalación comenzarían por la instalación de las dependencias, del interprete y librerías del proceso.

Listado B.1: Instalación básica para desarrollo.

```
apt-get install python3.8

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1
apk add --update --no-cache mariadb-dev jpeg-dev
apk add --update --no-cache --virtual .tmp-build-deps \
    gcc libc-dev linux-headers python3-dev musl-dev zlib zlib-dev
apk add openssl
apk add libffi-dev openssl-dev
apk add gcc musl-dev python3-dev libffi-dev openssl-dev cargo
env ARCHFLAGS="-arch x86\_64" LDFLAGS="--L/usr/local/opt/openssl/lib" \
        CFLAGS="--I/usr/local/opt/openssl/include"
python -m pip install --upgrade pip
pip install cryptography
pip install -r requirements.txt # Instalar conjunto de librerias

chmod +x ./manage.py
```

Las librerías del fichero de configuración requirements.txt son:

Listado B.2: Fichero con librerías Python.

```
Django
mysqlclient
python-dateutil
parse
mysql-connector
django-mysql
pymysql
mysql-client
setuptools
future
django-mathfilters
django-admin-list-filter-dropdown
openpyxl
PyMySQL
channels
djangorestframework
channels_redis
django-createsuperuserwithpassword
```

B.3 Instalación de Docker

La instalación de Docker dependerá del sistema operativo elegido:

B.3.1 Linux

Listado B.3: Lista de comandos para instalar Docker en Linux.

```
#Actualizacion del sistema
sudo apt update
sudo apt upgrade
#Paquetes previos necesarios
sudo apt-get install  curl apt-transport-https ca-certificates software-properties-common
#Repositorio Docker
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(
    lsb_release -cs) stable"
#Actualizar para recargar repositorio
sudo apt upgrade

#Instalacion de Docket
sudo apt install docker-ce
#Verificacion de estado
sudo systemctl status docker


#Docker de prueba
sudo docker run hello-world
```

B.3.2 En Windows

Es necesario tener habilitado la opción de HYPER-V en la características de Windows: Después realizar

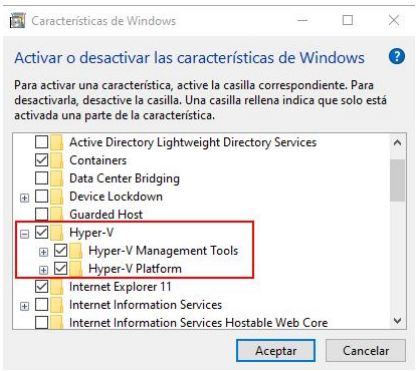

Figura B.1: Características de Windows.

la descarga desde la pagina web de Docker el programa e instalarlo. Es posible que requiera alguna configuración de acceso a carpetas dentro del mismo.

B.4 Preparación del contenedor Docker

Ejecutando los ficheros creados en el apartado de autocomposición Docker se encuentra los ficheros que hay que crear y los comandos básicos para su ejecución.

B.5 Auto instalación con CMD

Se ha creado unos scripts para su ejecución en Windows mediante CMD, podrían crearse los mismo en formato SSH para sistemas Linux.

- CrearDocker.cmd: Permite crear el entorno Docker.

  CrearDocker.sh: Versión Linux

- UpdateDocker.cmd: Permite actualizar relación del modelo con la base de datos.

  UpdateDocker.sh: Versión Linux

- UpDocker.cmd: Levantar el proceso Docker.

  UpDocker.sh: Versión Linux

**Los ejecutables cuentan con un lanzamiento de la página web.*

Apéndice C

Herramientas y recursos

Las herramientas necesarias para la elaboración del proyecto han sido:

- PC compatible
- Sistema operativo GNU/Linux [3]
- Procesador de textos Notepath++[4] o bloc de notas
- Interprete Python [5]
- (Ocasionalmente) Depurador lenguaje Python: Spyder[6]

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá