



# ENGINE GRAM

Proyecto de Final de Grado Superior de Desarrollo de  
Aplicaciones Multiplataforma.

## DESCRIPCIÓN BREVE.

En estas memorias se describirá paso a paso el proceso de desarrollo de la aplicación, así como datos de interés como tipo de software utilizado y su instalación o decisiones que se han ido tomando en el camino.

**Raúl Gutiérrez España.**

Alumno de 2º Grado en Cesur Cartuja.

## Índice

1. Resumen.....	1
2. Introducción.....	1
3. Objetivos y características del proyecto.....	3
3.1. FrontEnd.....	3
3.1.1. Inicio de Sesión.....	3
3.1.2 Administración de vehículos.....	3
3.2 Backend.....	3
3.2.1. Base de datos.....	4
3.2.2 ApiRest.....	4
3.2.3. Llamadas a la Api.....	5
4. Finalidad: Qué queremos conseguir con su implementación.....	5
5. Medios materiales usados: humanos, hardware, software.....	6
5.1 Humanos.....	6
5.2 Hardware.....	6
5.2 Software.....	6
5.2.1 Visual Studio Code.....	7
5.2.2 NODE JS.....	7
5.2.3 React Native.....	8
5.2.4 Expo CLI.....	8
5.2.5 Symfony CLI.....	8
5.2.6 PhpMyAdmin.....	9
5.2.7 XAMPP.....	9
6. Planificación del proyecto.....	9
6.1 Entorno Expo CLI.....	9
6.2 Entorno de Symfony.....	11
6.2.1Composer.....	11
6.2.2 Scoop.....	13
6.3 XAMPP.....	15
6.4 FrontEnd.....	15
6.4.1 Inicio de Sesión.....	16
6.4.2 Red Social.....	21
6.5 Backend.....	24
6.5.1 Base de datos.....	24

6.5.2 Api Symfony. ....	24
6.5.3 Funcionalidades de la Aplicación.....	27
7. Fase de pruebas .....	31
7.1 Inicio de sesión.....	31
7.2 Creación de Usuario.....	34
7.3 Listado de Vehículos.....	35
7.4 Creación de Vehículos. ....	36
8. Conclusiones y trabajos futuros o posibles mejoras.....	37
9. Referencias bibliográficas.....	37

## 1. Resumen

El problema que se ha tratado es la inexistencia de una *red social* para amantes del motor. Para ello, se ha creado una aplicación móvil, que es tanto para Android como para IOS en la cual se podrá mediante usuarios añadir vehículos y ver los de los demás.

Esto hace que la comunidad del motor esté más unida y se fomente la colección, el mantenimiento, la modificación, y el intercambio de vehículos a motor.

En la aplicación se creará una cuenta para poder acceder a ella. Una vez dentro para poder añadir un vehículo (que podrá ser coche o moto) deberá completar los datos necesarios en el formulario y subir una imagen. Una vez terminado ese proceso se podrá ver el vehículo añadido en la aplicación en la pantalla de visualización.

## 2. Introducción.

A la hora de crear una *red social* para la comunidad del motor lo quisimos hacer accesible y de fácil uso. Para ello decidimos crear una aplicación para los dos sistemas operativos del mercado, Android e IOS. Es decir, el código de la aplicación es *híbrido* y se puede desplegar en las dos plataformas de aplicaciones dependiendo el sistema.



La idea de la aplicación es tener una interfaz sencilla, que todo el mundo pueda usar con facilidad, pero que, a la vez, sea *útil* y *completa*.

Quisimos abarcar solamente los sectores de las  *motos*  y  *los coches*  para poder llegar a un público más específico y no intentar llegar a todo el público ya que pensamos que

llenar la red social de sectores que estuvieran vacíos o casi vacíos debido a la poca afluencia de usuarios de esas características, daría una imagen vacía o incompleta de nuestra idea original.



Todos los vehículos que se añaden son subidos a una *base de datos* global, por lo que cada vez que se entra en la aplicación con el usuario se podrá ver todos los vehículos hasta la fecha, ya sean nuevos creados a partir del inicio de sesión, o ya creados anteriormente. Esto hace que se convierta en una galería atemporal de vehículos de interés.



### 3. Objetivos y características del proyecto.

Los objetivos del proyecto se pueden categorizar en dos grandes secciones:

#### 3.1. FrontEnd.

En este apartado vamos a ver las partes en las que se compone la sección de frontend, que consiste en la parte visible que tiene el usuario una vez que accede a la aplicación, la interfaz para ser más concretos.

A la hora de diferenciar las partes de este podemos encontrar el inicio de sesión y la administración de vehículos una vez pasado esta primera parte.

##### 3.1.1. Inicio de Sesión.

Aquí podemos encontrar un conjunto de pantallas en las que principalmente podemos entrar con una cuenta ya creada que este en la base de datos o crear una nueva y registrarla.

A la hora de iniciar sesión se nos dará la opción de recordar la contraseña y de registrar una nueva cuenta dado que cabe la posibilidad que sea la primera vez que se entra en la aplicación o de que no tengamos ninguna cuenta registrada.

A la hora de recordar la contraseña tendremos que introducir nuestro correo electrónico de la cuenta en cuestión y se enviará un email con la contraseña de nuestra cuenta.

Si optamos por la opción de crear una nueva cuenta, se nos enviará a la pantalla para realizarlo.

##### 3.1.2 Administración de vehículos.

Aquí podemos encontrar principalmente dos pantallas.

En la primera pantalla podemos ver como todos los vehículos existentes en la base de datos serán visualizados de forma descendente mostrando todos los datos de interés.

En la segunda podremos observar una especie de formulario en el que tendremos que aportar los datos necesarios, así como una imagen para poder subir nuestro vehículo.

#### 3.2 Backend.

En este apartado veremos las partes en las que se dividen este proyecto en cuanto a *Backend*, que consiste en la parte de código y de funcionalidades que el usuario no ve desde la interfaz, es decir el usuario solo ve los elementos interactivos, pero no lo que estos hacen en segundo plano

### 3.2.1. Base de datos.

La base de datos que vamos a necesitar es una base de datos relacional para guardar los datos en tablas. Estos se harán desde *phpMyAdmin*.

En primer lugar, vamos a necesitar una tabla para poder administrar los usuarios. Esta será su estructura:

Tabla Usuarios:

- Nombre.
- Apellidos.
- Correo Electrónico.
- Nick.
- Contraseña.

Luego para poder administrar los vehículos que se suban desde la aplicación crearemos una tabla.

Tabla Vehículos:

- Marca.
- Modelo.
- Apodo.
- Matricula.
- Descripción.
- Fecha.
- Imagen.
- Tipo.

### 3.2.2 ApiRest.

Una ApiRest es la parte que comunica la aplicación con la base de datos, de forma que, el usuario simplemente tiene que rellenar formularios y clicar en elementos interactivos y el programador solamente tiene que escribir una sola vez la parte del código que se encarga de trabajar con la base de datos.

Todo esto funciona mediante *llamadas* al api, la cuales se envían con un cuerpo o *body* donde se encuentran diferentes parámetros necesarios, y se envían a una dirección con su ruta y parámetros correspondientes, que hace que, la Api lo reciba y procese la información.

La Api necesita un *mapeo* de todas las entidades que tenga la base de datos para así poder trabajar con esta. Además de esto necesita unos controladores que mediante unas

*rutas* se pueda decidir con cual entidad interactuar, así como, *métodos* para poder diferenciar el tipo de llamada que se realiza.

La Api para poder funcionar correctamente en el cuerpo se le tiene que entregar los datos en forma de *Json*, y a su vez esta te devolverá un cuerpo con el mismo formato o un cuerpo con un parámetro booleano.

### 3.2.3. Llamadas a la Api.

Esta parte es concreta, se trata de crear métodos que son llamados por los elementos colocados en la parte de FrontEnd que se encarguen que llamar a la Api y enviarles los datos necesarios en el formato necesario.

Así como llamarla para obtener los vehículos existentes y poder hacer que se muestren en la pantalla de visualización.

## 4. Finalidad: Qué queremos conseguir con su implementación.

A la hora de implementar nuestra aplicación nuestra intención es conseguir una visión del mundo del motor más amigable y cercana. Desde siempre ha sido un hobby un poco marginado en ese sentido y la comunidad ha tenido que resguardarse en foros cutres de internet que tenían poca o nula visibilidad.

Mediante nuestra app, uno mismo desde el store de su dispositivo móvil puede descargarla y usarla. Es un sistema bastante sencillo pero muy vistoso, simplemente con un sistema de cuentas de usuario convencional y un par de pantallas fáciles de usar conseguimos que todo esté más unido.

La aplicación tendrá unas funcionalidades de lanzamiento que serán ampliadas a medida que los usuarios nos entreguen un feedback y unas necesidades adicionales que no estén implementadas, haciendo que se convierta en una aplicación variable a gusto de los consumidores.



## 5. Medios materiales usados: humanos, hardware, software

### 5.1 Humanos.

Los medios humanos utilizados en este proyecto ha sido una única persona, el autor de este documento:

Raúl Gutiérrez España, estudiante de Cesur en el Grado Superior de Desarrollo de Aplicaciones Multiplataforma.

### 5.2 Hardware.

Para realizar la aplicación se ha necesitado únicamente mi ordenador personal, que está pensado para gaming por lo que no hemos tenido problemas de recursos a la hora de utilizar emuladores y multitud de programas en multitarea.



- Procesador: Intel i7 9700f 3GHZ
- Memoria RAM: 16GB
- Tarjeta Gráfica: NVIDIA RTX 3070 8GB
- Disco Duro 1: 500GB SSD
- Disco Duro 2: 1TB HDD
- Sistema Operativo: Windows 10 pro

### 5.2 Software

Para poder desarrollar correctamente hemos necesitado una serie de programas:

### 5.2.1 Visual Studio Code.

*Visual Studio Code* es un software de edición de texto con consola de comandos implementados. Mediante un solo programa es posible programar múltiples *lenguajes* debido a su gran capacidad de albergar modificaciones y *plug ins* que permiten desde cambiar la apariencia hasta crear cualquier tipo de proyecto con cualquier tipo de consola de comandos.

Dentro de este hemos instalado los *plug ins* básicos para poder editar código de *javascript* que es el código base del framework que hemos utilizado en la aplicación.

También hemos instalado los *plug ins* necesarios para poder editar código *php* ya que, la *Api* esta creada mediante *symfony* que es un *framework* de *php*, así como la base de datos que esta administrada mediante *phpMyAdmin*.



### 5.2.2 NODE JS.

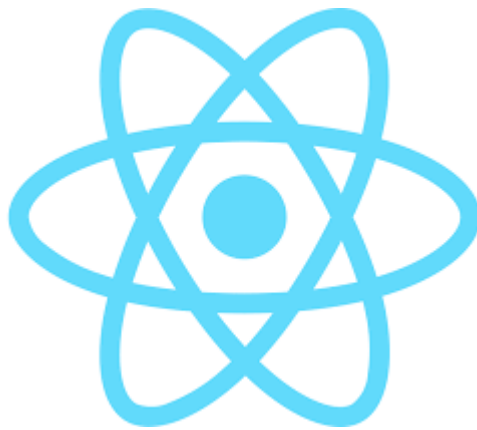
*Node Js* es una plataforma en la que podemos tener alojado nuestro código basado en javascript, lo que nos ahorrará mucho tiempo y muchos errores, de forma que no simplificará mucho la creación del proyecto que necesitamos para desarrollar la aplicación.



### 5.2.3 React Native.

*React Native* es una variante de React, se trata de un framework de JavaScript que nos permitirá crear aplicaciones móviles tanto para Android como para ios.

Utilizar elementos que pueden ser compilado en los dos sistemas operativos sin obtener resultados disonantes entre ellos, es decir, para cada elemento, existe un *homólogo* en cada sistema.



### 5.2.4 Expo CLI.

Para poder desarrollar en local y poder probar los avances que hagamos en la aplicación en React Native se ha decidido utilizar *Expo CLI* en local, debido a que React Native CLI debido a su sencillez de instalación y uso, ya que este último suele arrojar muchos errores en su mayoría laberínticos y con soluciones poco intuitivas y a priori no tiene ninguna desventaja en cuanto a nuestros objetivos en el desarrollo de la App.

Cabe destacar que se ha realizado en local para poder hacer pruebas con la Api que hemos creado sin tener que desplegarla en un servidor en la nube.

### 5.2.5 Symfony CLI.

*Symfony Cli* es una consola de comandos que nos permite generar proyectos en php que nos servirán para crear Apis personalizadas.

Nos permite crear un servidor en local donde alojar la Api para poder hacerle llamadas y probar que funciona.



#### 5.2.6 PhpMyAdmin.

Es un gestor de bases de datos que nos permite crear bases de datos y administrar su contenido. Es algo básico pero muy útil. De esta forma aprovechamos que usamos todos los servidores de prueba en php y no hay que instalar muchas más aplicaciones.



#### 5.2.7 XAMPP.

Es un administrador de servicios php el cual nos será de ayuda para poder utilizar servidores en local basados en php y poder probar las funcionalidades de nuestra aplicación como si estuviera desplegada en servidores online.



### 6. Planificación del proyecto

A continuación, se va a detallar todo el proceso del proyecto, desde la instalación del software hasta la creación de la aplicación:

#### 6.1 Entorno Expo CLI.

Como ya se ha explicado en un capítulo anterior lo que es Expo CLI, se procederá a la explicación de su instalación.

Es algo bastante sencillo y rápido.

Primero vamos a la página oficial de React Native, una vez allí buscaremos donde dice Get Started, y avanzaremos hasta el apartado *Setting up the development environment*

Una vez ahí seguiremos los pasos:

Instalamos mediante comandos en el cmd el CLI de expo.

```
npm install -g expo-cli
```

Luego creamos un proyecto expo con este comando.

```
expo init AwesomeProject
```

Luego de hacer todo esto tendremos una estructura de carpetas como esta:

.expo	29/04/2022 9:39	Carpeta de archivos	
.expo-shared	18/04/2022 11:50	Carpeta de archivos	
.git	09/05/2022 13:13	Carpeta de archivos	
api	09/05/2022 9:19	Carpeta de archivos	
assets	25/04/2022 11:03	Carpeta de archivos	
Memoria	11/05/2022 9:29	Carpeta de archivos	
node_modules	25/04/2022 13:21	Carpeta de archivos	
src	19/04/2022 11:48	Carpeta de archivos	
.gitignore	18/04/2022 11:50	Documento de te...	1 KB
App	19/04/2022 12:01	Archivo JavaScript	2 KB
app.json	25/04/2022 13:23	Archivo JSON	1 KB
babel.config	18/04/2022 11:50	Archivo JavaScript	1 KB
package.json	25/04/2022 13:21	Archivo JSON	2 KB
package-lock.json	25/04/2022 13:21	Archivo JSON	712 KB

Es aquí donde trabajaremos en la aplicación, para probar la aplicación tenemos que situarnos justo a esa altura del directorio y correr este comando:

```
npm start # you can also use: expo start
```

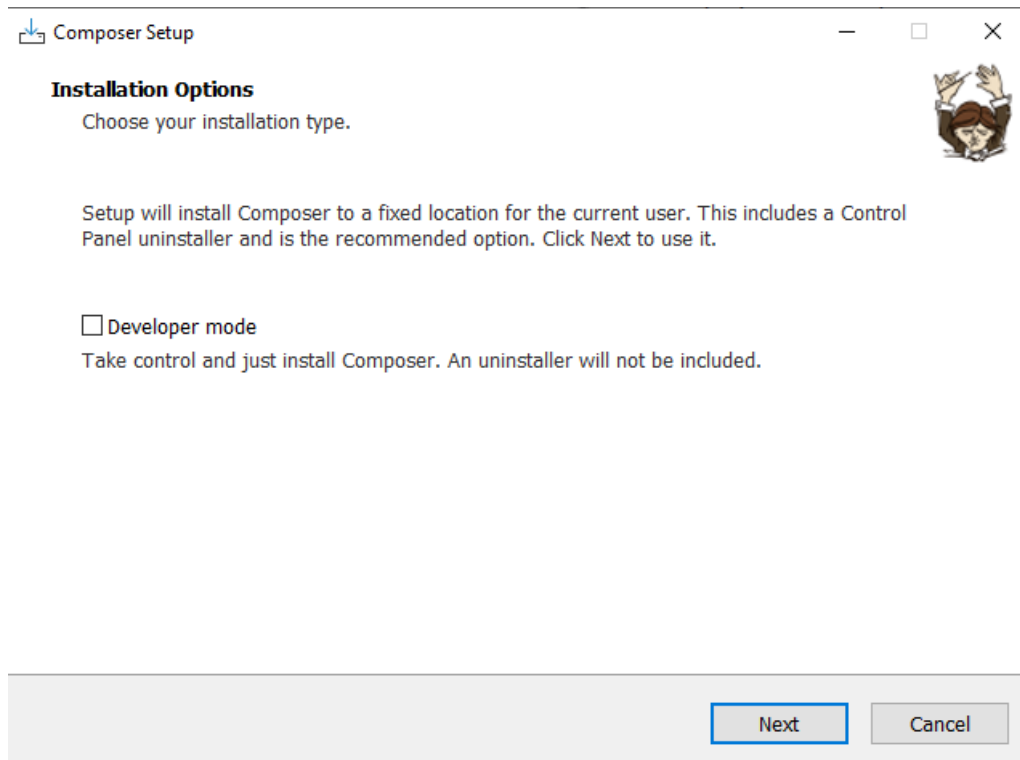
## 6.2 Entorno de Symfony.

Este es el segundo entorno que tenemos que instalar y es parecido al anterior:

### 6.2.1 Composer

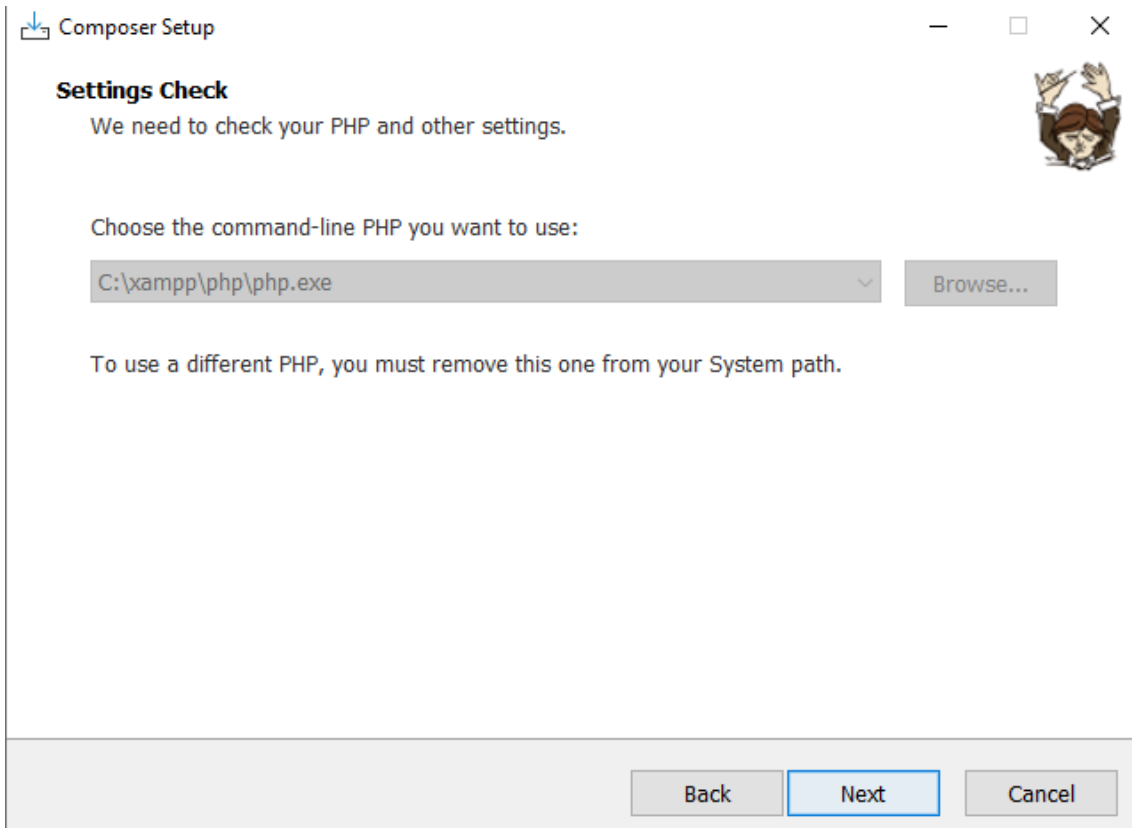
El primer paso es instalar *Composer* para poder instalar el entorno de symfony y poder utilizar sus comandos.

Para ello necesitamos tener PHP instalado cosa que con la aplicación XAMP que tenemos instalados nos sirve.

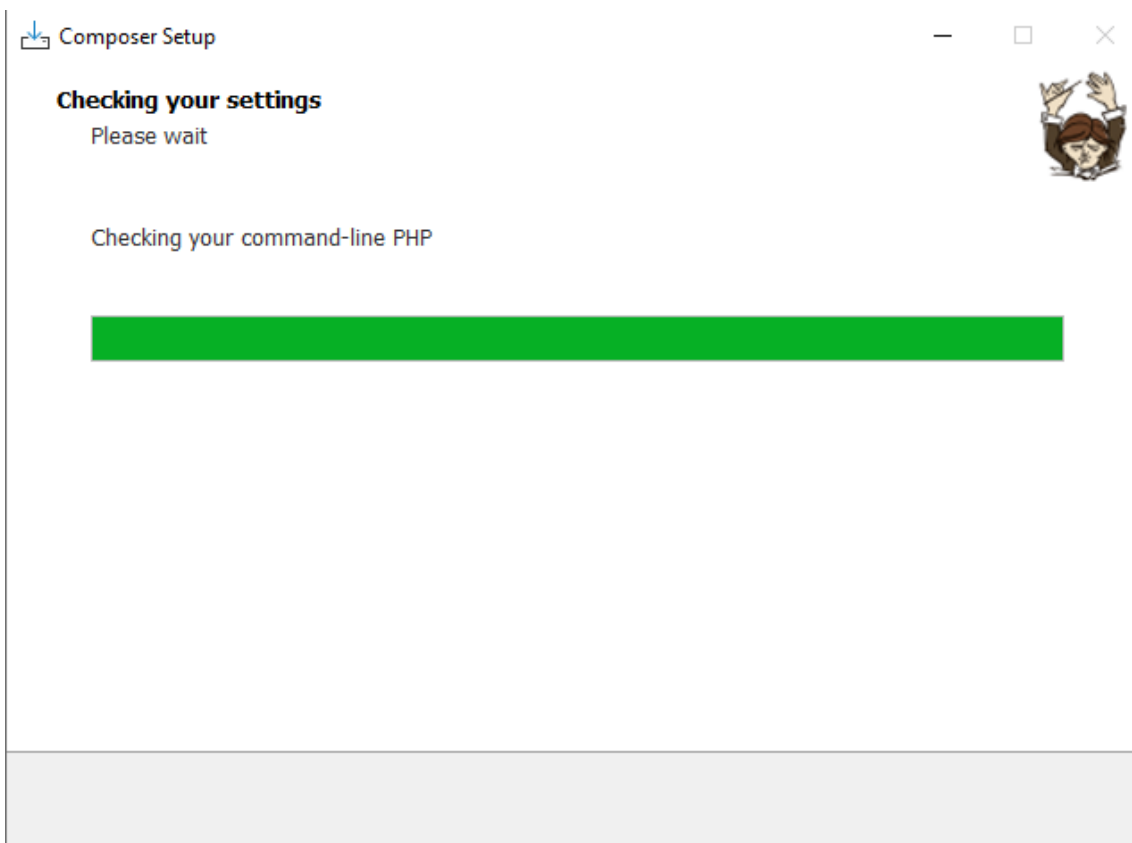


Necesitamos descargar Composer para luego ejecutar el instalador.

Next.



Next.



Y una vez que acabe, ya tendremos *composer* instalado, para comprobarlo podemos escribir en la consola de comandos el comando *composer*.

```
Composer version 2.2.9 2022-03-15 22:13:37

Usage:
  command [options] [arguments]

Options:
  -h, --help                Display this help message
  -q, --quiet               Do not output any message
  -V, --version             Display this application version
  --ansi                   Force ANSI output
  --no-ansi                Disable ANSI output
  -n, --no-interaction     Do not ask any interactive question
  --profile                Display timing and memory usage information
  --no-plugins             Whether to disable plugins.
  --no-scripts             Skips the execution of all scripts defined in composer.json file.
  -d, --working-dir=WORKING-DIR If specified, use the given directory as working directory.
  --no-cache               Prevent use of the cache
  -v|vv|vvv, --verbose    Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
```

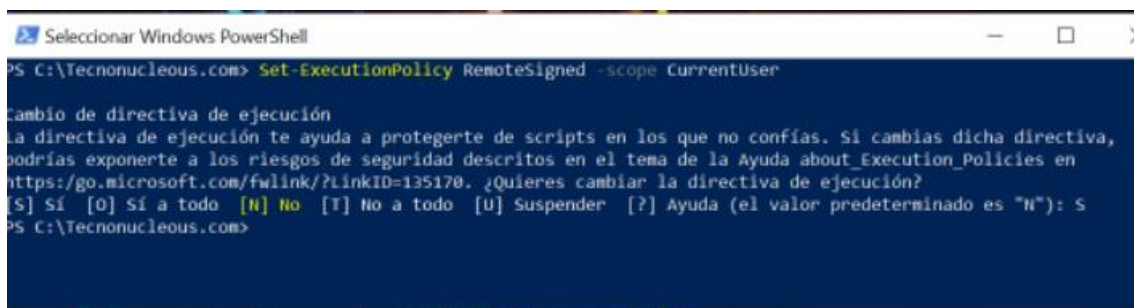
### 6.2.2 Scoop.

El segundo paso de este entorno es instalar *Scoop*, para ello necesitamos tener en el sistema la versión 5 o superior de PowerShell y .NET framework 4.5 o superior.

Una vez tengamos eso tenemos que ejecutar dos comandos, uno detrás de otro.

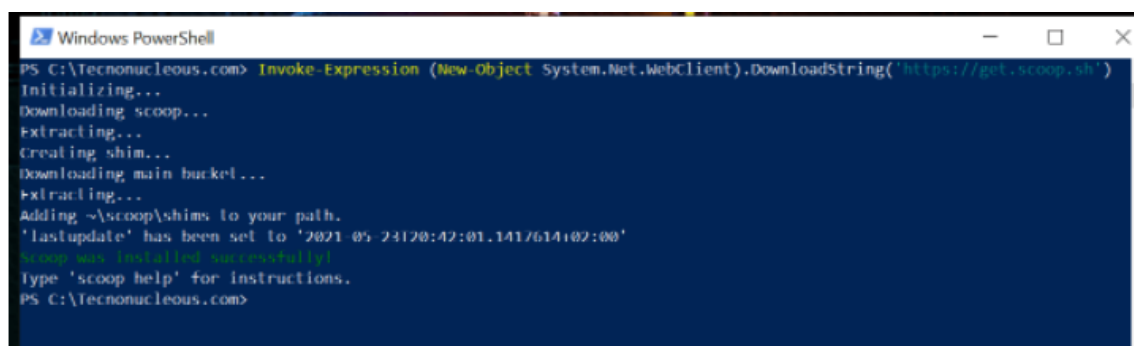
```
Set-ExecutionPolicy RemoteSigned -scope CurrentUser
```

```
Invoke-Expression (New-Object System.Net.WebClient).DownloadString('https://get.scoop.sh')
```



```
PS C:\Tecnonucleous.com> Set-ExecutionPolicy RemoteSigned -scope CurrentUser

Cambio de directiva de ejecución
La directiva de ejecución te ayuda a protegerte de scripts en los que no confías. Si cambias dicha directiva,
podrías exponerte a los riesgos de seguridad descritos en el tema de la Ayuda about_Execution_Policies en
https://go.microsoft.com/fwlink/?LinkID=135170. ¿Quieres cambiar la directiva de ejecución?
[S] Sí [O] Sí a todo [N] No [T] No a todo [U] Suspender [?] Ayuda (el valor predeterminado es "N"): S
PS C:\Tecnonucleous.com>
```



```
PS C:\Tecnonucleous.com> Invoke-Expression (New-Object System.Net.WebClient).DownloadString('https://get.scoop.sh')
Initializing...
Downloading scoop...
Extracting...
Creating shim...
Downloading main bucket...
Extracting...
Adding ~\scoop\shims to your path.
'lastupdate' has been set to '2021-03-24T20:42:01.1417614+02:00'
scoop was installed successfully!
Type 'scoop help' for instructions.
PS C:\Tecnonucleous.com>
```



Listo con esto podemos proseguir a instalar el CLI de symfony que como hemos dicho antes es similar a Expo CLI.

Ejecutamos el siguiente comando.

```
$ scoop install symfony-cli
```

Esto nos habilitara los comandos para poder usar Symfony, para a continuación, crear ya nuestro propio proyecto, la Api.

```
$ symfony new my_project
```

Esto nos generará un sistema de carpetas como este:

bin	09/05/2022 9:19	Carpeta de archivos	
config	09/05/2022 9:19	Carpeta de archivos	
public	09/05/2022 9:19	Carpeta de archivos	
src	09/05/2022 9:19	Carpeta de archivos	
.env	09/05/2022 9:19	Archivo ENV	1 KB
.gitignore	09/05/2022 9:19	Documento de te...	1 KB
composer.json	09/05/2022 9:19	Archivo JSON	2 KB
composer.lock	09/05/2022 9:19	Archivo LOCK	89 KB
symfony.lock	09/05/2022 9:19	Archivo LOCK	4 KB

Y para probar que funciona nos colocamos en esta altura del directorio y escribimos en la consola de comandos:

```
Windows PowerShell
PS C:\git\ProyectoFinDeCurso\api> symfony server:start
```

Y obtendremos algo como esto:

```
[OK] Web server listening
The web server is using PHP CGI 7.4.28
http://127.0.0.1:8000

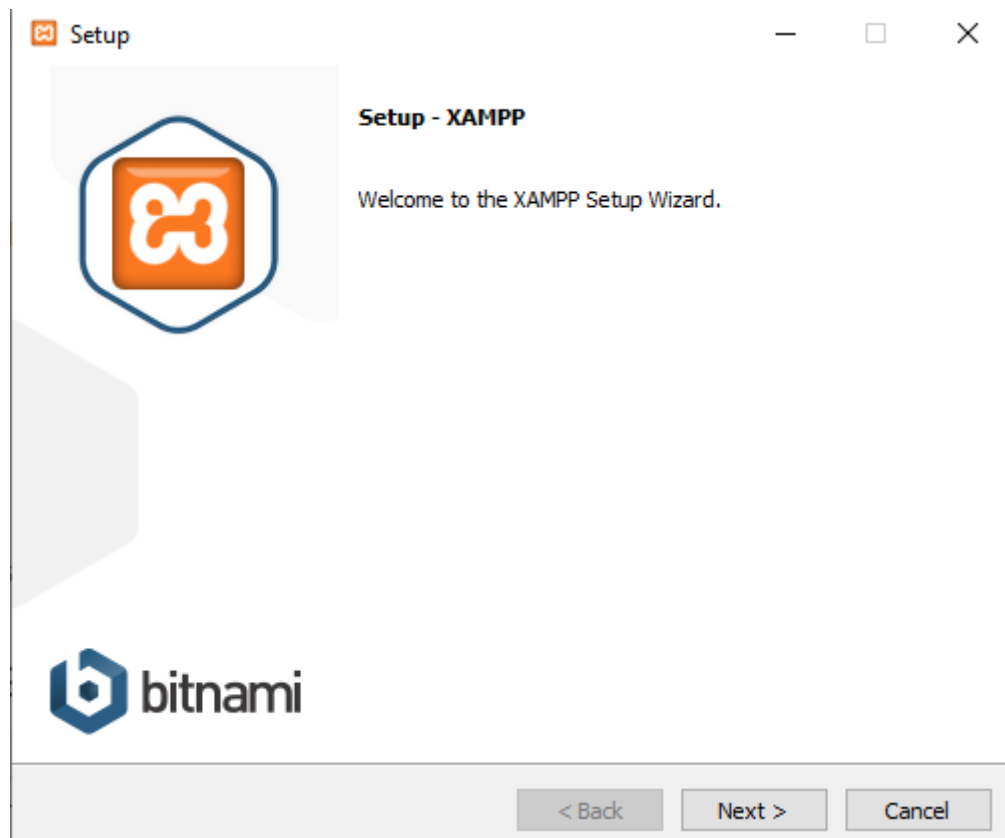
[Web Server ] May 11 10:53:10 | DEBUG | PHP | Reloading PHP versions
[Web Server ] May 11 10:53:11 | DEBUG | PHP | Using PHP version 7.4.28 (from default version in $PATH)
[Web Server ] May 11 10:53:11 | INFO | PHP | listening path="c:\\xampp\\php\\php-cgi.exe" php="7.4.28" port=60204
```

### 6.3 XAMPP.

Por último, nos quedaría instalar XAMPP por si no lo tuviéramos instalado ya, aunque es bastante corto.



Elegimos nuestra descarga.



Solo queda seguir el asistente de instalación y ya tendremos nuestro XAMPP instalado.

### 6.4 FrontEnd.

Una vez que tengamos todo lo necesario instalado, solo queda empezar con el desarrollo, para ello lo primero que se va a crear es la parte visual de cara al usuario, es decir el *FrontEnd*.

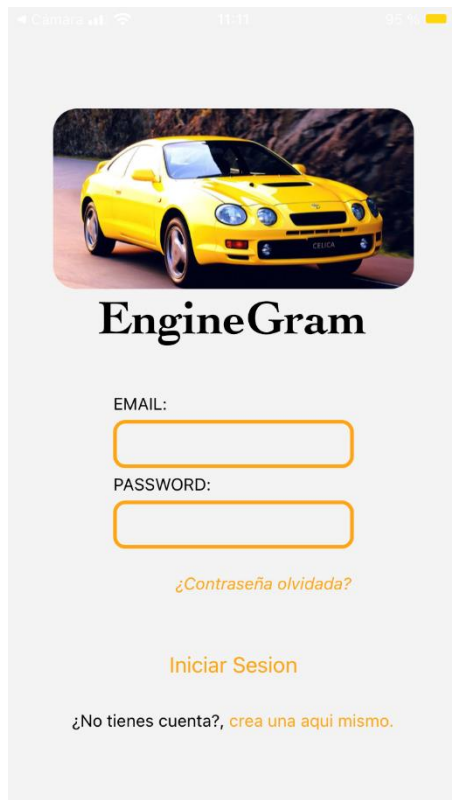
#### 6.4.1 Inicio de Sesión.

La primera parte del desarrollo del *FrontEnd* es el inicio de sesión.

Está compuesto por 3 pantallas dentro de un Stack.

```
<Stack.Navigator initialRouteName="IniciarSesion">
  <Stack.Screen options={{ headerShown: false }} name="IniciarSesion" component={IniciarSesionScreen} />
  <Stack.Screen options={{ headerShown: false }} name="CrearCuenta" component={CrearCuentaScreen} />
  <Stack.Screen options={{ headerShown: false }} name="ContraseñaOlvidada" component={ContraseñaOlvidadaScreen} />
  <Stack.Screen options={{ headerShown: false }} name="Main" component={MainStack} />
</Stack.Navigator>
```

En la que encontramos:



La página principal la cual está compuesta por dos `TextInput`s para introducir el correo y la contraseña.

```
<TextInput
  style={{
    height: 40,
    width: 200,
    marginBottom: 5,
    borderWidth: 3,
    borderColor: "orange",
    borderRadius: 10,
    padding: 10,
  }}
  onChangeText={onChangeEmailLogIn}
  value={emailLogIn}
```

```

<TextInput
  style={{
    height: 40,
    width: 200,

    borderWidth: 3,
    borderColor: "orange",
    borderRadius: 10,
    padding: 10,
  }}
  onChangeText={onChangePasswordLogIn}
  value={passwordLogIn}
/>

```

Luego, tenemos un texto interactivo para la contraseña, es decir al clicar en el texto, se nos llevara a la página para poder recuperar la contraseña.

```

<Text
  onPress={() => navigation.navigate('ContraseñaOlvidada')}
  style={{
    color: "orange",
    marginLeft: 50,
    marginTop: 20,
    fontStyle: "italic",
    marginBottom: 40
  }}>¿Contraseña olvidada?</Text>

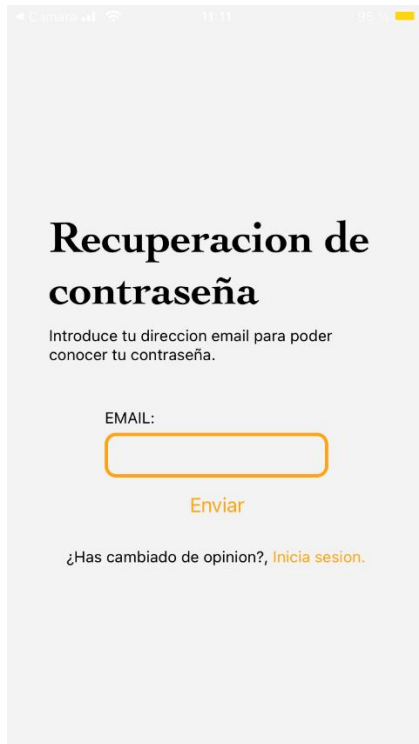
```

Encontramos también el botón de iniciar sesión que una vez que se pulse hace las comprobaciones sobre las variables modificadas en los TextInputs y si son correctas navegará al siguiente Stack que es el que muestra las pantallas de dentro de la aplicación.

Justo debajo tenemos un texto interactivo como el de la contraseña que nos lleva a la pantalla de Crear Cuenta.

```
<Text style={{ marginTop: 20 }}>
  ¿No tienes cuenta?,
  <Text onPress={() => navigation.navigate('CrearCuenta')} style={{ color: "orange" }}>crea una aqui mismo.
</Text>
</Text>
```

A continuación, tenemos la pantalla de recuperar contraseña.



Aquí tenemos un TextInput igual al de la pantalla anterior para obtener el email.

```
<TextInput
  style={{
    height: 40,
    width: 200,
    marginBottom: 5,
    borderWidth: 3,
    borderColor: "orange",
    borderRadius: 10,
    padding: 10,
  }}
  onChangeText={onChangeEmailLogIn}
  value={emailLogIn}
```

El botón principal se encarga de buscar el email en la base de datos, y una vez que lo encuentre, manda un correo electrónico a esa dirección con la contraseña correspondiente. En el caso de que no encuentre la dirección de correo electrónico en la base de datos arrojará un mensaje.

Por último, encontramos al final un texto interactivo que nos devuelve a la página de inicio de sesión.

```
<Text style={{ marginTop: 20 }}>
  ¿Has cambiado de opinion?,
  <Text onPress={() => navigation.navigate('IniciarSesion')} style={{ color: "orange" }}>Inicia sesion.
</Text>
</Text>
```

En la última pantalla de este Stack tenemos la de crear una cuenta nueva.

Aquí tenemos todos los InputTexts necesarios para recopilar todos los datos, de la misma forma que en las otras pantallas.

```

<Text style={{ marginBottom: 5 }}>NOMBRE:</Text>
<TextInput
  style={{
    height: 40,
    width: 200,
    marginBottom: 5,
    borderWidth: 3,
    borderColor: "orange",
    borderRadius: 10,
    padding: 10,
  }}
  onChangeText={onChangeNombreRegistro}
  value={nombreRegistro}
/>

```

El botón que encontramos para crear cuenta hace una llamada a la Api enviándoles los datos para que este realice un insert en la base de datos.

Y por último tenemos un texto interactivo que nos devuelve a la página principal de iniciar sesión.

```

<Text style={{ marginTop: 20 }}>¿Ya tienes cuenta?,
  <Text onPress={() => navigation.navigate('IniciarSesion')} style={{ color: "orange" }}>Inicia sesion.
</Text>
</Text>

```

### 6.4.2 Red Social.



Esta es la Pagina de visualización de Vehículos, se trata de un Tab Stack que contiene dos tabs, uno para visualizar y otro para crear.

En el de visualizar que es este, consiste en un flatlist que coge los datos de un Json que llega desde la Api con una consulta con todos los vehículos de la base de datos.

```
return (  
  <SafeAreaView >  
    <FlatList  
      data={datos}  
      renderItem={renderItem}  
      keyExtractor={item => item.id}  
    />  
  </SafeAreaView >  
);
```



```
<Image
  style={{ width: 300, height: 175 }}
  source={require('../assets/coche.jpg')}
/>
```

Así es como renderizamos la imagen.

```
<Text style={{fontSize:30, marginBottom:20}}>{item.apodo}</Text>
<View>
  <Text style={{fontSize:20}}>{item.marca} {item.modelo}</Text>
  <Text><Text style={{fontSize:20}}>Matricula:</Text> {item.matricula}</Text>
  <Text><Text style={{fontSize:20}}>Fecha de matriculacion:</Text> {item.fecha}</Text>
  <Text><Text style={{fontSize:20}}>Tipo de vehiculo:</Text> {item.tipo}</Text>
  <Text><Text style={{fontSize:20}}>Descripcion:</Text> {"\n"}{item.descripcion}</Text>
</View>
```

Y así es como renderizamos los datos del vehículo.

Si pulsamos en el otro Tab de la aplicación nos lleva a la pantalla de creación de vehículos.

Consta de una pantalla que recolecta datos mediante TextInputs, y dos Pickers, los textInputs son iguales a los anteriormente mostrados, pero los Pickers son cada uno diferentes.

```

<Picker
  selectedValue={Tipo}
  onValueChange={({itemValue, itemIndex}) =>
    setTipo(itemValue)
  }
  style={{
    width: 350,
    height: 10
  }}

  <Picker.Item label="Coche" value="Coche" />
  <Picker.Item label="Moto" value="Moto" />
</Picker>

```

Este es el Picker de Tipos, es simplemente un elemento que clicando te deja elegir varias opciones, en este caso hemos creado dos categorías solamente, motos y coches.

```

<Button
  title="Seleccionar imagen"
  onPress={() => selectImage()}
  color="orange"
  style={{marginLeft:20}}
/>

```

Este es el botón de seleccionar imagen que nos lleva a un método que abre un menú para elegir una foto de la galería del dispositivo.

```

const selectImage = async () => {
  let result = await ImagePicker.launchImageLibraryAsync({
    mediaTypes: ImagePicker.MediaTypeOptions.All,
    allowsEditing: true,
    aspect: [4, 3],
    quality: 1,
  });
  console.log(result);
  if (!result.cancelled) {
    setImage(result.uri);
  }
};

```

Este método abre un menú en el que eliges una foto y guarda sus datos para posteriormente poder trabajar con ella.

Y al final posee un botón que se encarga de procesar los datos y realizar la llamada a la Api para introducirlos en la base de datos.

## 6.5 Backend

Una vez que hayamos creado la parte visible para el usuario, ahora, nos queda programar todas las interacciones que van *ocultas* desde la interfaz.

Comenzamos desde lo más básico, la base de datos.

### 6.5.1 Base de datos

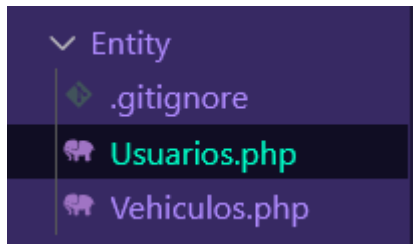
Como hemos dicho antes la base de datos esta subida en un servidor local de php(xampp) y para crear las tablas que necesitamos tenemos este código:

```
create table usuarios(  
    id_usuario int primary key,  
    nombre varchar(250),  
    apellidos varchar (250),  
    correo varchar(250),  
    nick varchar(250),  
    contraseña varchar(250)  
  
)  
|  
  
create table Vehiculos(  
id_vehiculo int primary key AUTO_INCREMENT,  
    marca varchar(250),  
    modelo varchar(250),  
    apodo varchar(250),  
    matricula varchar(250),  
    descripcion varchar (1000),  
    tipo varchar(250)  
);
```

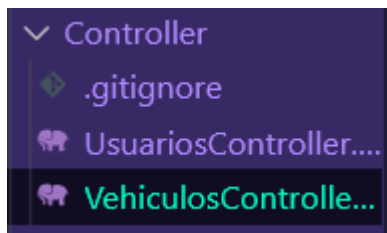
En esta fase en la tabla de vehículos no incluimos el apartado de *imagen* debido a que será una funcionalidad futura. (ver capítulo 8).

### 6.5.2 Api Symfony.

Una vez que creamos la base de datos con las tablas necesarias, comenzamos a desarrollar la Api:



Primero clonamos las entidades(tablas) de la base de datos que hemos creado antes, para posteriormente crear los controllers de cada una.



Los controladores están compuestos por una serie de elementos, los cuales son:

```
/**
 * @Route("/Vehiculos",name="app_vehiculo", methods={"GET","POST","PUT","DELETE"})
 */
```

La ruta.

```
function postMethod($request, $doctrine): string
```

Y los 4 diferentes métodos para cada tipo de llamada POST, GET, PUT, DELETE.

```
if ($request->getMethod() == "POST") {
    return new Response($this->postMethod($request, $doctrine));
} else if ($request->getMethod() == "GET") {
    return $this->getMethod($doctrine);
} else if ($request->getMethod() == "PUT") {
    return new Response($this->putMethod($request, $doctrine));
} else if ($request->getMethod() == "DELETE") {
```

Controlado en el código de esta manera, de forma que se puede utilizar una sola ruta para los 4 tipos de llamadas posibles.

```
// /**
//  * @return Usuarios[] Returns an array of Usuarios objects
//  */
public function Consulta(): array
{
    $conn = $this->getEntityManager()->getConnection();

    $sql = '
        SELECT * FROM USUARIOS
    ';
    $stmt = $conn->prepare($sql);
    $resultSet = $stmt->executeQuery();

    return $resultSet->fetchAllAssociative();
}
```

Este es el método al que llama el GetMethod (uno de los antes mostrado).

```
$usuario = new Usuarios();
$usuario->setNombre($UsuarioInfo->nombre);
$usuario->setApellidos($UsuarioInfo->apellidos);
$usuario->setCorreo($UsuarioInfo->correo);
$usuario->setNick($UsuarioInfo->nick);
$usuario->setContraseña($UsuarioInfo->contrasena);

$entityManager->persist($usuario);
$entityManager->flush();
```

Y así es como se introduce un nuevo usuario.

Estos son los dos métodos principales que se van a utilizar, aunque están operativos los otros dos (put y delete) para posibles mejoras a futuro.

```
$usuario = $doctrine->getRepository(Usuarios::class)->find($UsuarioInfo->id);
if ($usuario != null) {
    $usuario->setNombre($UsuarioInfo->nombre);
    $usuario->setApellidos($UsuarioInfo->apellidos);
    $usuario->setCorreo($UsuarioInfo->correo);
    $usuario->setNick($UsuarioInfo->nick);
    $usuario->setContraseña($UsuarioInfo->contraseña);

    $entityManager->flush();
    $endValue = true;
}
```

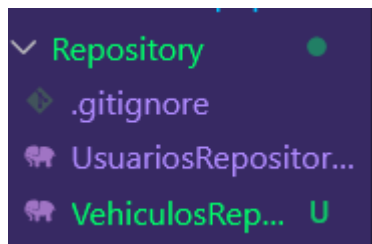
PUT.

```
try {
  $usuario = $doctrine->getRepository(Usuarios::class)->find($UsuarioInfo->id);
  $entityManager->remove($usuario);
  $entityManager->flush();
  $endValue = true;
}
```

DELETE.

El proceso es el mismo para los dos controladores, entidades y repositorios.

Otro elemento son los repositorios de cada entidad (ya mencionados antes).



### 6.5.3 Funcionalidades de la Aplicación.

Comenzamos en el orden en el que fuimos cuando se explicó la parte de FrontEnd.

Empezamos con el Inicio de sesión.

```
</View>
<Button
  title="Iniciar Sesion"
  onPress={() => inicioSesion()}
  color="orange"
/>
```

Como vemos el botón nos lleva al método que realiza el proceso:

```
function inicioSesion() {
```

```
  if (emailLogIn == null || passwordLogIn == null) {
    Alert.alert('rellena todos los campos')
```

Primero comprobamos que los datos introducidos no son nulos.

```

} else {
  for (var i = 0; i < usuariosBox.length; i++) {

```

Si no es nulo recorreremos el Json obtenido de la base de datos con los datos de los usuarios.

```

const obtenerUsuarios =()=>{
  let url='http://127.0.0.1:8000/Usuarios';
  fetch(url)
  .then(response=>response.json())
  .then((responseJson)=>{
    setUsuariosBox(responseJson)
  })
}

```

Así es como se obtienen.

```

if (emailLogIn == usuariosBox[i].correo) {
  if (passwordLogIn == usuariosBox[i].contraseña) {
    bandera = true;
    navigation.navigate('Main');
    break;
  }
  else {
    bandera=false;
  }
} else {
  bandera=false;
}
}

```

A continuación, comprobamos que la combinación de datos introducidos coincide en los datos, y vamos jugando con una bandera(boolean) para poder controlar errores.

```

if (bandera === false) {
  Alert.alert('Los datos introducidos no son coincidentes');
  bandera = 0;
} else {
  Alert.alert('Sesion iniciada correctamente');
}

```

Finalmente controlamos un mensaje de salida dependiendo de la bandera, ahí su utilidad.

Seguimos con la creación de usuarios.

```
fetch('http://127.0.0.1:8000/Usuarios', {  
  method: 'POST',
```

En el método al que llamamos desde el botón de crear usuario realizamos una llamada de tipo POST a la Api.

```
body: JSON.stringify({  
  "nombre": nombreRegistro,  
  "apellidos": ApellidosRegistro,  
  "correo": emailRegistro,  
  "nick": NickRegistro,  
  "contrasena": passwordRegistro  
}),
```

En ella especificamos el cuerpo, el cual será un JSON con los datos obtenidos al introducirlos en los TextInput.

```
.then((responseJson) => {  
  console.log('getting data from fetch', responseJson);  
  Alert.alert("Usuario Creado Correctamente");  
})  
.catch(error => console.log(error))
```

Y finalmente controlamos el mensaje del usuario y los errores.

*Cabe aclarar antes de seguir para la red social que la funcionalidad de la contraseña no está todavía implementada por ser una funcionalidad pensada para futuras mejoras.*

Seguimos con el listado de usuarios.

```
<SafeAreaView >  
  <FlatList  
    data={datos}  
    renderItem={renderItem}  
    keyExtractor={item => item.id}  
  />  
</SafeAreaView >
```

La pantalla consiste en un flatlist que carga un método renderItem.

```
function renderItem({ item }) {  
  return /
```



```

<Image
  style={{ width: 300, height: 175 }}
  source={require('../assets/coche.jpg')}
/>
<Text style={{fontSize:30, marginBottom:20}}>{item.apodo}</Text>
<View>
  <Text style={{fontSize:20}}>{item.marca} {item.modelo}</Text>
  <Text><Text style={{fontSize:20}}>Matricula:</Text> {item.matricula}</Text>
  <Text><Text style={{fontSize:20}}>Fecha de matriculacion:</Text> {item.fecha}</Text>
  <Text><Text style={{fontSize:20}}>Tipo de vehiculo:</Text> {item.tipo}</Text>
  <Text><Text style={{fontSize:20}}>Descripcion:</Text> {"\n"}{item.descripcion}</Text>
</View>

```

```

const obtenerVehiculos =()=>{
  let url='http://127.0.0.1:8000/VehiculosApi';
  fetch(url)
    .then(response=>response.json())
    .then((responseJson)=>{
      setDatos(responseJson)
    })
}

```

Y así es como obtenemos los datos de los vehículos.

*Hay que aclarar que la función de la fotografía no está todavía implementada debido a ser una mejora futura en la aplicación.*

Por último, tenemos la pantalla de crear vehículos.

Simplemente desde el botón de crear vehículos llamamos a la Api.

```

fetch('http://127.0.0.1:8000/VehiculosApi', {
  method: 'POST',
  body: JSON.stringify({
    "marca": marcavehiculo,
    "modelo": modeloVehiculo,
    "apodo": apodoVehiculo,
    "matricula": matriculaVehiculo,
    "descripcion": DescripcionVehiculo,
    "tipo": tipoVehiculo
  }),

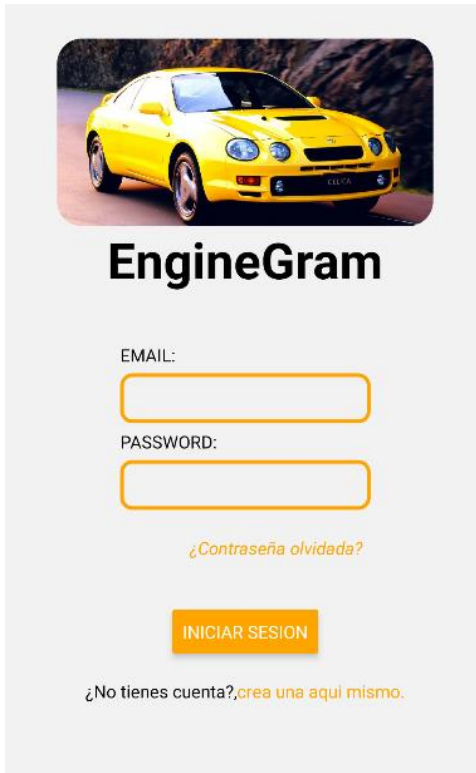
```

Casi idéntico a la llamada creada para crear un nuevo usuario.

## 7. Fase de pruebas

En este capítulo se pondrá a prueba las funcionalidades de la aplicación, así como su aspecto final.

### 7.1 Inicio de sesión.



The image shows a login page for 'EngineGram'. At the top, there is a photograph of a yellow sports car. Below the photo, the text 'EngineGram' is displayed in a large, bold, black font. Underneath the name, there are two input fields: one for 'EMAIL:' and one for 'PASSWORD:'. Both fields have orange borders. Below the password field, there is a link that says '¿Contraseña olvidada?'. At the bottom of the form area, there is an orange button with the text 'INICIAR SESION'. Below the button, there is a link that says '¿No tienes cuenta?, crea una aquí mismo.'.

**EngineGram**

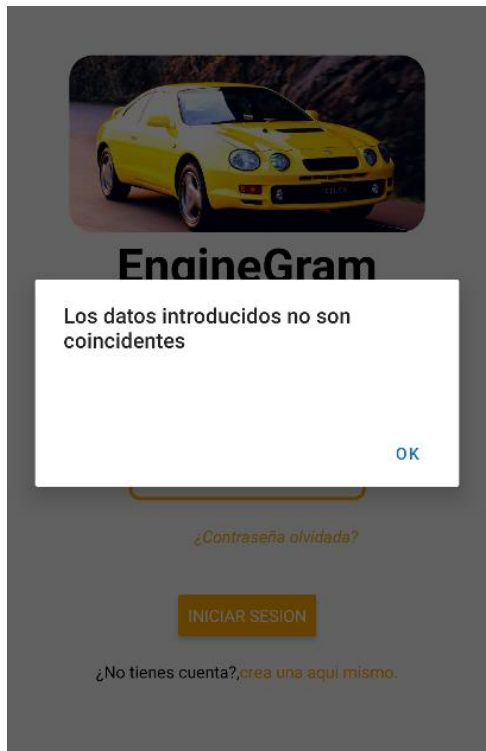
EMAIL:

PASSWORD:

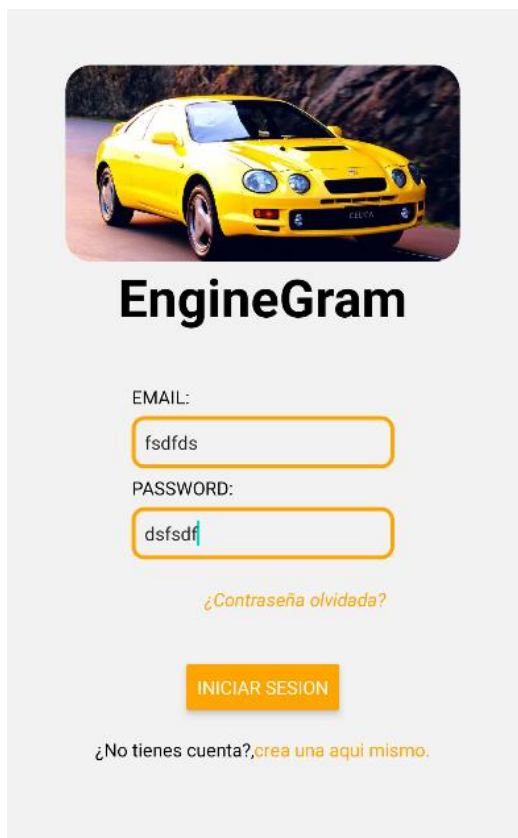
[¿Contraseña olvidada?](#)

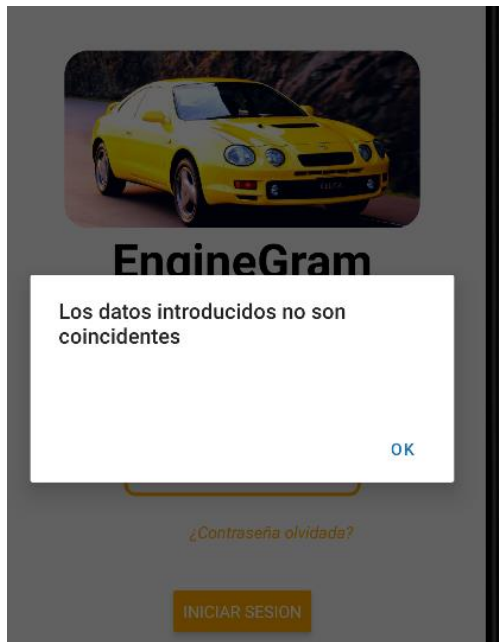
**INICIAR SESION**

[¿No tienes cuenta?, crea una aquí mismo.](#)



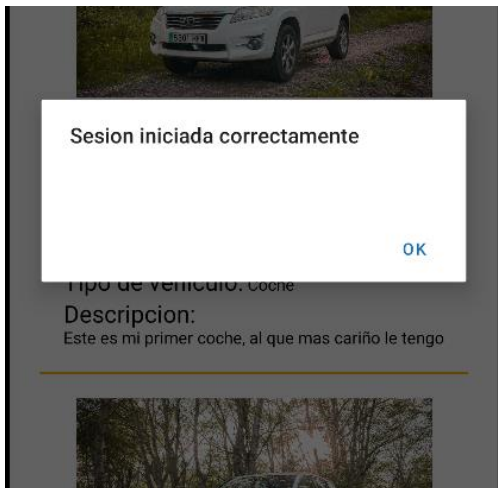
Si dejamos los huecos en blanco obtenemos este mensaje.





Si no introducimos unos datos que existan obtenemos este mensaje.

A screenshot of the EngineGram login page. At the top is a yellow sports car. Below it, the text "EngineGram" is displayed. There are two input fields: "EMAIL:" with the text "root" and "PASSWORD:" with the text "root". Below the password field is a link that says "¿Contraseña olvidada?". At the bottom is a button labeled "INICIAR SESION" and a link that says "¿No tienes cuenta?, crea una aquí mismo."



Finalmente, si introducimos los datos correctos entraremos en la parte de red social y obtendremos este mensaje.

## 7.2 Creación de Usuario.

# Crear Cuenta

NOMBRE:

APELLIDOS:

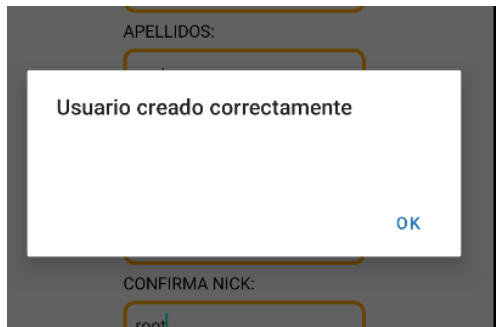
EMAIL:

CONTRASEÑA:

CONFIRMA NICK:

[CREAR CUENTA](#)

¿Ya tienes cuenta? [Inicia sesion.](#)




A la hora de crear usuario es sencillo de ver, nos crea un usuario con los datos que le hemos introducido.

### 7.3 Listado de Vehículos.



Dentro de la red social tenemos el listado de Vehículos, que es renderizado por un JSON y cómo podemos ver se muestran todas las propiedades de los vehículos.

#### 7.4 Creación de Vehículos.



CrearVehiculos

Rellena los datos de tu vehiculo

Citroen c4

c4trin 0105GPC

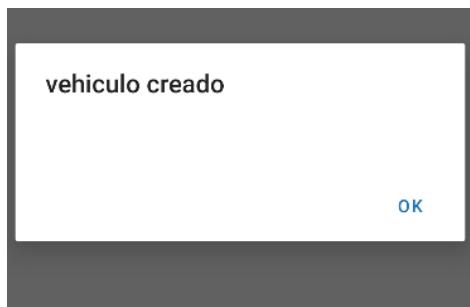
Descripcion de mi coche

2021/12/5

SELECCIONAR IMAGEN

Coche

El formulario tiene un encabezado naranja con el título 'CrearVehiculos'. El cuerpo principal es gris claro y contiene el título 'Rellena los datos de tu vehiculo'. Hay dos filas de campos de texto con los valores 'Citroen c4' y 'c4trin 0105GPC'. Debajo hay un campo de texto para la descripción, un campo de fecha con el valor '2021/12/5', un botón naranja 'SELECCIONAR IMAGEN' y un menú desplegable con el valor 'Coche'. La barra de navegación inferior tiene un fondo dividido en naranja y rojo, con un icono de coche en naranja y un icono de lista en rojo.



Al darle al botón de crear creara un vehículo con los atributos introducidos.

## 8. Conclusiones y trabajos futuros o posibles mejoras

La versión de la aplicación alcanzada es suficiente para ser una primera versión de prueba, es funcional, aunque limitada.

Por ejemplo, las funcionalidades de contraseña y de imágenes en el listado de vehículos no están implementadas debido a que, por un lado:

Tenemos que investigar más sobre la administración de correos electrónicos desde aplicaciones que se envían desde ciertas plataformas online.

Y por otro tenemos que investigar más acerca de la codificación de archivos binarios como las imágenes en una BBDD tanto como para cargar como descargar, por lo que ahora mismo nos resulta bastante complejo.

Cabría destacar que en algunos formularios no se ha controlado todas las opciones que el usuario pudiera introducir, ya que a falta de modificar la base de datos debido a mejoras futuras y tratarse de una versión de prueba no se ha visto eficiente controlar todo eso.

## 9. Referencias bibliográficas.

Para poder realizar este proyecto se ha utilizado y consultado material principalmente de estos sitios web:

-<https://reactnative.dev/docs/getting-started>

-<https://symfony.com/doc/current/index.html>

-<https://stackoverflow.com/>