



Django

Turbo Cargado

Desmitificando,
Heredando
y Potenciando



- Soy ingeniero en Software. 
- Apasionado de GNU/Linux y el software libre.  
- Amante de la montaña y ciclista ocasional.  
- Autodidacta de corazón. 
- Creo firmemente que Python es genial  
- y que puedes expandir tus horizontes con Django.   



Hola, soy Emilio!



Contenido

1

Desmitificando el Modelo de Usuario de Django

2

Explorando la Herencia de los modelos de Django

3

Potenciando el Sitio de Administración



01

Desmitificando el Modelo de Usuario de Django



Lo que no sabías del modelo de usuario de Django



Campo de Usuario Sensible a Mayúsculas:

- **Comparación con Vehículos:** Es como si una clave de tu auto funcionara de manera diferente si usas mayúsculas o minúsculas.
- **Problema:** Los usuarios "juandelospalotes" y "Juandelospalotes" son considerados diferentes. ¡Eso no es bueno!
- **Soluciones:** Usa campos "case-insensitive" o ajusta tu backend de autenticación.



Lo que no sabías del modelo de usuario de Django



Campo de Usuario con Validación de Letras Unicode:

- **Comparación con Vehículos:** Imagina que tu carro solo acepta ciertos tipos de gasolina, pero no otros.
- **Problema:** Django acepta caracteres especiales en nombres de usuario, lo que puede causar confusión.
- **Soluciones:** Usa validadores específicos o ajusta las reglas de validación de nombres de usuario.



Lo que no sabías del modelo de usuario de Django



Email no Es Único:

- **Comparación con Vehículos:** Sería como si varias personas compartieran la misma matrícula de auto.
- **Problema:** Múltiples usuarios pueden usar la misma dirección de correo, lo que puede causar problemas en la recuperación de contraseñas.
- **Soluciones:** Redefine el modelo de usuario o valida el correo en tus formularios.



Lo que no sabías del modelo de usuario de Django



Email no Obligatorio:

- **Comparación con Vehículos:** Sería como si algunos autos no tuvieran un sistema de encendido.
- **Problema:** Los usuarios pueden omitir proporcionar una dirección de correo, lo que podría ser un problema para la autenticación.
- **Soluciones:** Establece un correo obligatorio o ajusta tus formularios.



Lo que no sabías del modelo de usuario de Django



Cambiar el Modelo de Usuario Después de Comenzar es Difícil:

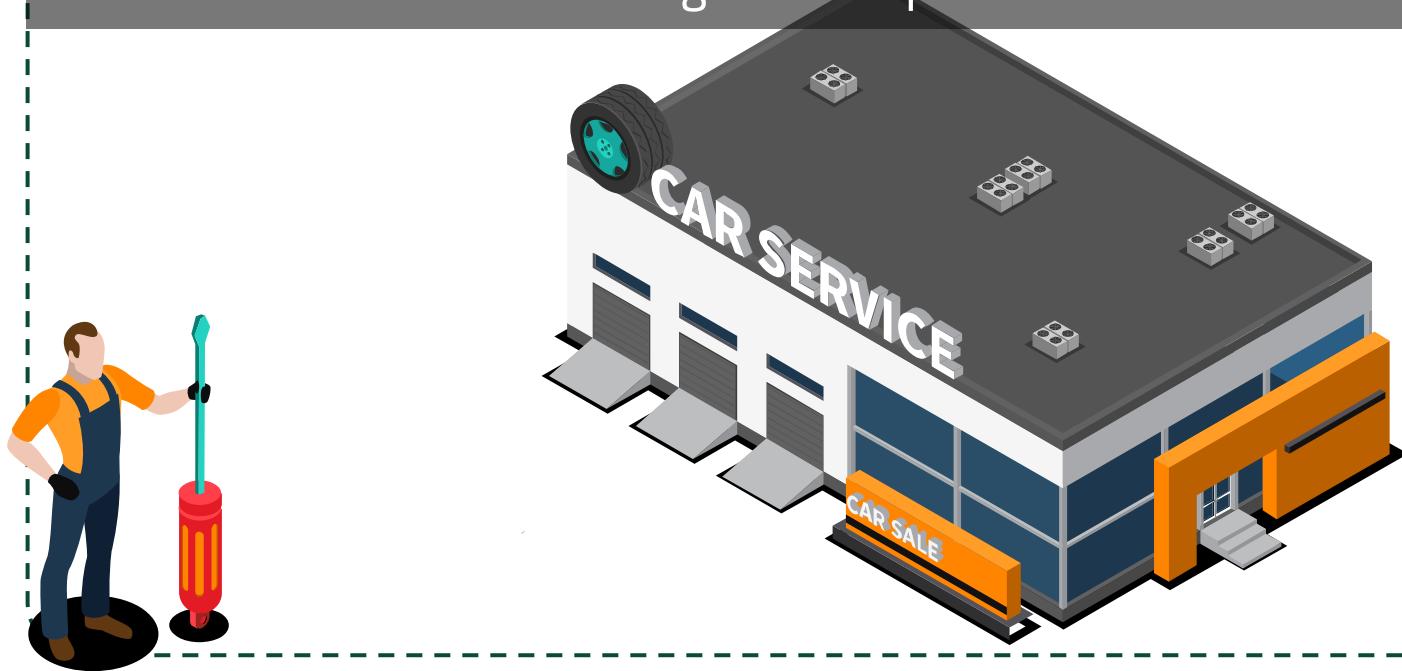
- **Comparación con Vehículos:** Cambiar la transmisión después de construir todo el auto es complicado.
- **Problema:** Cambiar el modelo de usuario más tarde puede causar problemas en la base de datos y referencias.
- **Soluciones:** Siempre ajusta el modelo de usuario desde el principio para evitar dolores de cabeza futuros.





Soluciones a los Problemas del Modelo de Usuario en Django:

Arregla tu Máquina en Marcha





Trabajos de Mantenimiento:

- **Comparación con Vehículos:** Imagina que tienes un auto antiguo que necesita algunas reparaciones para funcionar correctamente.
- **Enfoque:** Si ya tienes un proyecto en producción que utiliza el modelo de usuario predeterminado de Django, es mejor aplicar soluciones temporales.





Reemplazo Completo:

- **Comparación con Vehículos:** Esto sería como cambiar todo el motor de tu carro por uno nuevo y mejorado desde el principio.
- **Enfoque:** Si estás comenzando un proyecto Django desde cero, elige esta opción para evitar problemas futuros.





Soluciones Temporales:

- Si no hay nombres de usuario conflictivos, puedes tomar medidas:
 - En tus formularios de registro, evita que los nombres de usuario conflictivos creen cuentas.
 - En tu formulario de creación de usuario, utiliza la validación para el nombre de usuario.





Haciendo el Campo de Usuario Insensible a Mayúsculas:

```
nvim user_case_insen ~/D/C/django_turbo_cargado
```

```
8 def clean_username(self):
7     username = self.cleaned_data.get("username")
6     if User.objects.filter(username__iexact=username).exists():
5         self.add_error(
4             "username",
3             "Ya existe un usuario con este nombre de usuario."
2         )
1     return username
9
```

INSERT <ser_case_insensitive.py i < 41 Bot 9:1 17:20





Cambiar la Validación de Nombre de Usuario para Usar Solo Letras ASCII:



A screenshot of a terminal window showing code being edited in nvim. The terminal title is "nvim user_ASCII_vali ~/D/C/django_turbo_cargado". The code defines a custom UsernameField class that adds an ASCIIUsernameValidator to its validators.

```
nvim user_ASCII_vali ~/D/C/django_turbo_cargado
from django.contrib.auth.forms import UsernameField
from django.contrib.auth.validators import ASCIIUsernameValidator

class ASCIIUsernameField(UsernameField):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.validators.append(ASCIIUsernameValidator())


```

The terminal status bar shows "NORMAL" mode, the file name "user_ASCII_valitdation.py", and the current line number "41".



Cambiar la Validación de Nombre de Usuario para Usar Solo Letras ASCII:



A screenshot of a terminal window showing code in nvim. The title bar says "nvim user_creation_f ~/D/C/django_turbo_cargado". The code is a Python class definition for a UserCreationForm:

```
7 class UserCreationForm(forms.ModelForm):
6     # field definitions...
5
4     class Meta:
3         model = User
2         fields = ("username",)
1         field_classes = {'username': ASCIIUsernameField}
```

The status bar at the bottom shows "NORMAL" and the file name "user_creation_form.py". It also displays some navigation and status information: ⌂ 41, Bot, 8:1, ⌂ 18:32.



Hacer el Correo Electrónico único y Obligatorio:



```
nvim email_unique.py ~/D/C/django_turbo_cargado

17 class UserCreationForm(forms.ModelForm):
16     email = forms.EmailField(required=True)
15     # other field definitions...
14
13     class Meta:
12         model = User
11         fields = ("username",)
10         field_classes = {'username': ASCIIUsernameField}
9
8     def clean_email(self):
7         email = self.cleaned_data.get("email")
6         if User.objects.filter(email__iexact=email).exists():
5             self.add_error(
4                 "email",
3                 _("Ya existe un usuario con este correo electrónico."))
2
1         return email
18
```

INSERT ➤ email_unique.py 39 < 41 Bot 18:1 18:55



Reemplazo Completo:

Si estás comenzando un proyecto desde cero, aquí hay algunos consejos para evitar los problemas desde el principio:

- Cambia el modelo de usuario predeterminado.
- Personaliza los formularios de registro para asegurarte de que el correo electrónico sea único y obligatorio desde el principio.





Reemplazar el Modelo de Usuario Predeterminado en Django: Haciendo tu Propia Máquina a Medida





Dos Enfoques para Reemplazar el Modelo de Usuario:

- **Extender AbstractUser:** Es útil cuando deseas modificar los métodos existentes, agregar campos adicionales o cambiar el administrador de objetos.
- **Extender AbstractBaseUser:** Esta opción te brinda un control total sobre el modelo de usuario, lo que significa que puedes eliminar campos o cambiar su definición. Sin embargo, ten en cuenta que esta opción es más complicada.





Extendiendo desde AbstractBaseUser:

```
nvim abstract_base_u ~/D/C/django_turbo_cargado
 1  from django.contrib.auth.base_user import AbstractBaseUser
 2  from django.contrib.auth.models import PermissionsMixin, UserManager
 3  from django.contrib.auth.validators import ASCIIUsernameValidator
 4  from django.contrib.postgres.fields import CICharField, CIEmailField
 5  from django.core.mail import send_mail
 6  from django.db import models
 7  from django.utils import timezone
 8  from django.utils.translation import gettext_lazy as _
 9
10 class CustomUser(AbstractBaseUser, PermissionsMixin):
11     # Validador para el nombre de usuario (ASCII)
12     username_validator = ASCIIUsernameValidator()
13
14     # Campo de nombre de usuario (insensible a mayúsculas y minúsculas)
15     username = CICharField(
16         _("nombre de usuario"),
17         max_length=150,
18         unique=True,
19         help_text=_("Requerido. 150 caracteres o menos. Letras, dígitos y @./+/-/_ solamente."),
20         validators=[username_validator],
21         error_messages={
22             "unique": _("Un usuario con este nombre de usuario ya existe."),
23         },
24     )
25
26     # Campo de primer nombre
27     first_name = models.CharField(_("primer nombre"), max_length=150, blank=True)
28
29     # Campo de apellido
30     last_name = models.CharField(_("apellido"), max_length=150, blank=True)
31
```



Extendiendo desde AbstractBaseUser:

```
nvim abstract_base_u ~/D/C/django_turbo_cargado
  4     email = CIEmailField(
  5         _("dirección de correo electrónico"),
  6         unique=True,
  7         error_messages={
  8             "unique": _("Un usuario con esta dirección de correo electrónico ya existe."),
  9         }
 10    )
 11
 12    # Campo para determinar si el usuario es parte del personal (staff)
 13    is_staff = models.BooleanField(
 14        _("estatus de personal"),
 15        default=False,
 16        help_text=_("Indica si el usuario puede iniciar sesión en este sitio de administración."),
 17    )
 18
 19    # Campo para determinar si el usuario está activo
 20    is_active = models.BooleanField(
 21        _("activo"),
 22        default=True,
 23        help_text=_(
 24            "Indica si este usuario debe considerarse activo. Desactive esto en lugar de eliminar cuentas."
 25        ),
 26
 27
 28    # Campo para la fecha de registro del usuario
 29    date_joined = models.DateTimeField(_("fecha de registro"), default=timezone.now)
 30
 31    objects = UserManager()
 32
 33    EMAIL_FIELD = "email" # Campo para la dirección de correo electrónico
 34    USERNAME_FIELD = "username" # Campo para el nombre de usuario
 35    REQUIRED_FIELDS = ["email"] # Campos requeridos durante la creación de usuario
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
```



Extendiendo desde AbstractBaseUser:

```
nvim abstract_base_u ~D/C/django_turbo_cargado
 5
 4
 3
 2
 1
72 class Meta:
    verbose_name = _("usuario") # Nombre singular del modelo
    verbose_name_plural = _("usuarios") # Nombre plural del modelo
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
 12
 13
 14
 15
 16
 17
def clean(self):
    super().clean()
    self.email = self._class_.objects.normalize_email(self.email)

def get_full_name(self):
    """
    Devuelve el primer nombre y el apellido, separados por un espacio.
    """
    nombre_completo = "%s %s" % (self.first_name, self.last_name)
    return nombre_completo.strip()

def get_short_name(self):
    """
    Devuelve el nombre corto del usuario.
    """
    return self.first_name

def email_user(self, subject, message, from_email=None, **kwargs):
    """
    Envía un correo electrónico a este usuario.
    """
    send_mail(subject, message, from_email, [self.email], **kwargs)
```





02

Explorando la Herencia de los modelos de Django

¿Por Qué Django es la Familia Perfecta para tu
Aplicación?



Tipo de herencias en modelos de Django

1. **Clases base abstractas:** Utilizadas cuando la clase padre contiene campos comunes, pero no deseas que se cree una tabla separada para la clase padre en la base de datos.
2. **Herencia de múltiples tablas:** Utilizada cuando la clase padre tiene campos comunes y deseas que se cree una tabla independiente para la clase padre en la base de datos.
3. **Modelos proxy:** Utilizados cuando deseas modificar el comportamiento de la clase padre, como cambiar el orden o agregar un nuevo administrador de modelos.





Clases Base Abstractas

```
nvim herencia_clase_ ~/D/C/django_turbo_cargado
17 from django.db import models
16
15 class AbstractPerson(models.Model):
14     first_name = models.CharField(max_length=50)
13     last_name = models.CharField(max_length=50)
12     email = models.EmailField()
11
10     class Meta:
9         abstract = True
8
7 class Employee(AbstractPerson):
6     job_title = models.CharField(max_length=100)
5     salary = models.IntegerField()
4
3 class Customer(AbstractPerson):
2     address = models.CharField(max_length=200)
1     phone_number = models.CharField(max_length=20)
18
```

NORMAL ➤ herencia_clase_abstracta.py ⏷ 41 Bot 18:1 ⏶ 13:19





Herencia de Múltiples Tablas

```
nvim herencia_multip ~/D/C/django_turbo_cargado
15 from django.db import models
14
13
12 class Person(models.Model):
11     first_name = models.CharField(max_length=50)
10     last_name = models.CharField(max_length=50)
9     email = models.EmailField()
8
7 class Employee(Person):
6     job_title = models.CharField(max_length=100)
5     salary = models.IntegerField()
4
3 class Customer(Person):
2     address = models.CharField(max_length=200)
1     phone_number = models.CharField(max_length=20)
16
```

NORMAL ><ncia_multiples_tablas.py 13j < ⊞ 41 ◀ Bot 16:1 ◀ ⊙ 15:04





Herencia modelo proxy

```
 0 ● ●  nvim herencia_modelo ~/D/C/django_turbo_cargado
20 from django.db import models
19
18 class Person(models.Model):
17     first_name = models.CharField(max_length=50)
16     last_name = models.CharField(max_length=50)
15     email = models.EmailField()
14
13 class Employee(Person):
12     job_title = models.CharField(max_length=100)
11     salary = models.IntegerField()
10
9
8 class Manager(Employee):
7     class Meta:
6         proxy = True
5
4     def get_employees(self):
3         return Employee.objects.filter(
2             job_title='Manager', salary__gt=self.salary
1
21
```

NORMAL <> cia_modelo_proxy.py gj < ⊞ 41 Bot 21:1 ① 15:28





03

Potenciando el Sitio de Administración:

De 0 a Superhéroe en Django Admin





Sitio de administración de Django

The screenshot shows the Django administration interface at localhost:8000/admin/. The top navigation bar includes links for Site administration | Django, local host:8000/admin/, WELCOME, ADMIN@EXAMPLE.COM, VIEW SITE / CHANGE PASSWORD / LOG OUT, and a user icon.

The main content area is titled "Site administration". It features several horizontal sections:

- AUTHENTICATION AND AUTHORIZATION**: Contains a "Groups" entry with "Add" and "Change" buttons.
- CATALOG**: Contains a "Catalogs" entry with "Add" and "Change" buttons.
- PEOPLE**: Contains "Customers", "Employees", and "People" entries, each with "Add" and "Change" buttons.
- USERS**: Contains a "Users" entry with "Add" and "Change" buttons.
- VEHICLES**: Contains "Makes", "Models", and "Vehicles" entries, each with "Add" and "Change" buttons.

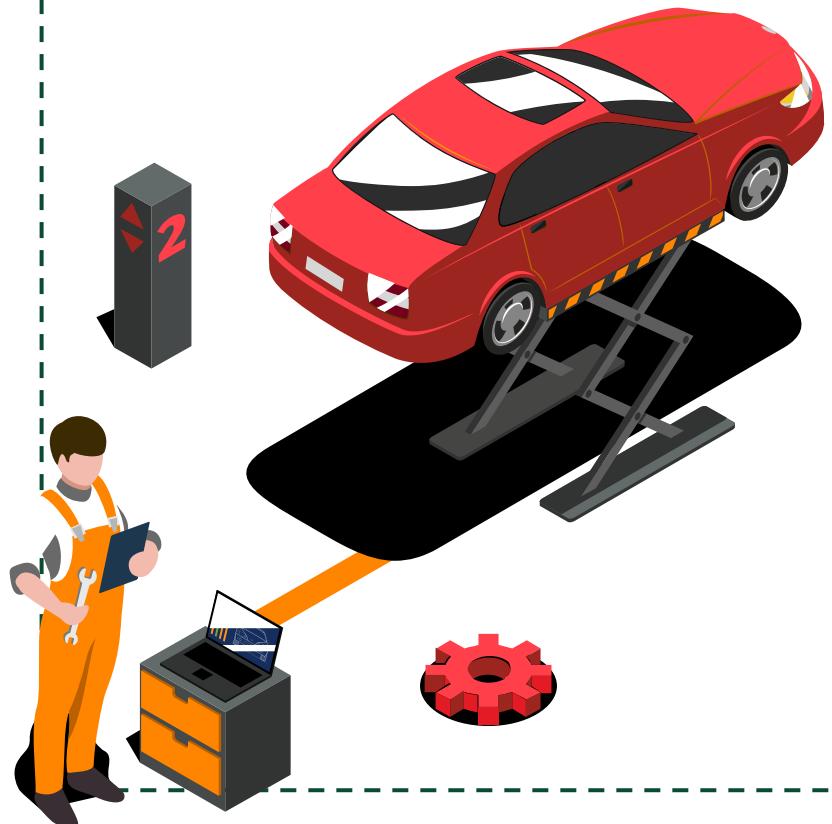
To the right, there are two sidebar sections:

- Recent actions**: A list of recent actions, mostly related to vehicle objects and catalog entries.
- My actions**: A list of actions taken by the user, specifically related to Toyota makes.



Django Jazzmin

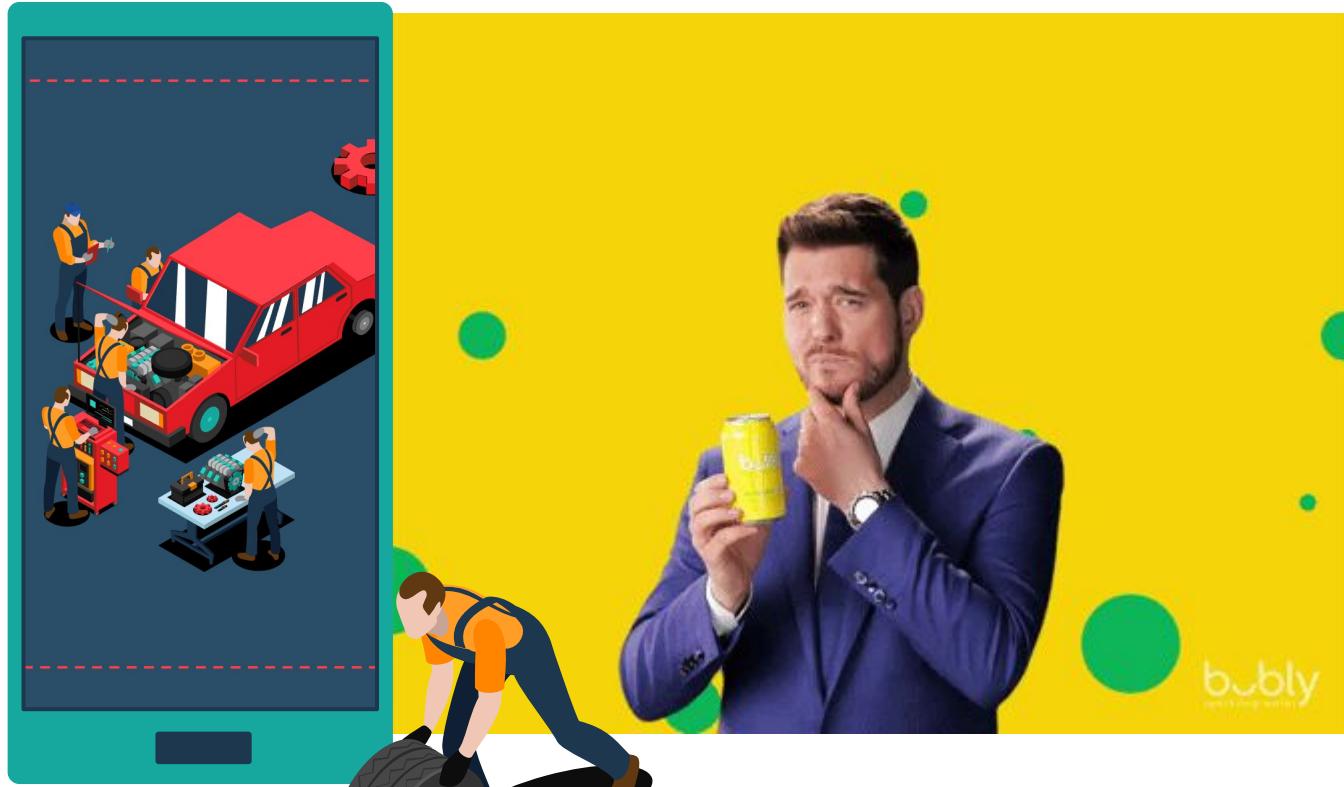
The screenshot shows the Django Jazzmin admin interface for a project named "DJ Car Services". The top navigation bar includes tabs for "Site administration | Django" and "Site administration | DJ Car Services". The main dashboard has a dark sidebar on the left containing links for "Dashboard", "Authentication and Authorization", "Groups", "People" (with "Customers" and "Employees"), "Catalog" (with "Catalogs"), "Users", and "Vehicles". The main content area is divided into three columns: "Authentication and Authorization" (containing "Groups" with "Add" and "Change" buttons), "Catalog" (containing "Catalogs" with "Add" and "Change" buttons), "People" (containing "Customers", "Employees", and "People" with "Add" and "Change" buttons), "Users" (containing "Users" with "Add" and "Change" buttons), and "Vehicles" (containing "Makes", "Models", and "Vehicles" with "Add" and "Change" buttons). A "Recent actions" sidebar on the right lists three entries: "Vehicle object (1)" added 8 hours, 47 minutes ago, "Added 'Vehicle object (1)'", and two entries for "Cambio de aceite de motor" added 9 hours, 40 minutes ago and 9 hours, 41 minutes ago.



Gracias!

Tienes alguna pregunta?



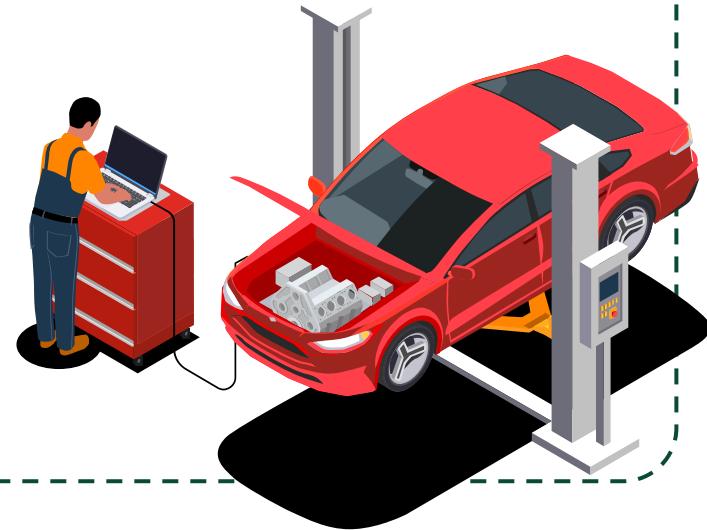




Recursos



<https://bit.ly/3QrkUqU>





Emilio Ferreyra

Django/Python Developer
emilioferreyra@gmail.com

Emilio Ferreyra

Backend Developer | Django/Python | GNU/Linux | Git | Docker

Santiago, Santiago, Dominican Republic · [Contact info](#)

919 followers · 500+ connections



<https://bit.ly/40wpIQB>



