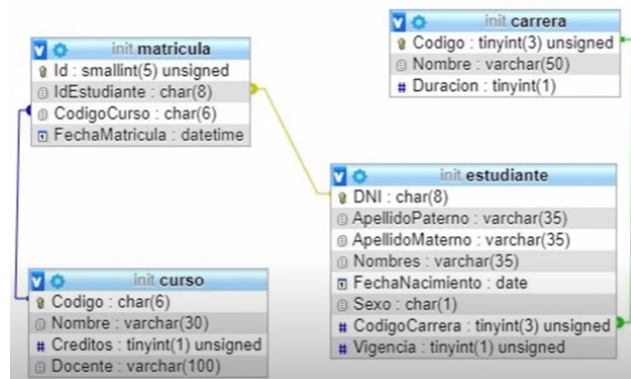




# Proyecto Django Para Administrar Universidad

## Aplicación Web con Base de Datos y Envío de Correos

### Esquema de Base de Datos



### Instalamos Django desde la terminal

```
#Verificamos que tengamos Python instalado con:
$ python --version
#pip(Package Installer Python) es un comando y herramienta y tenemos que ver que la tengamos
$ pip
#Instalamos Django en djangoproject.com con:
$ pip install Django==3.2.5
```

Documentación de Django: <https://docs.djangoproject.com/en/3.2/>

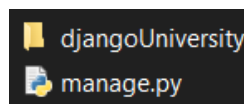
## Creación del proyecto

Una vez creada la carpeta que alojará al proyecto, ejecutamos el primer comando de Django para la creación de este.

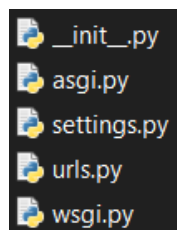
```
$ django-admin startproject djangoUniversity
```

Tenemos dos resultantes:

Manage es el archivo mas importante, nos permitirá crear la base de datos, usuarios y migraciones para una base de datos real. Nos permite iniciar un pequeño servidor para desplegar las aplicaciones web.



La carpeta: contiene



init: indica que la carpeta va a ser tratada como un paquete de Python

asgi(Asynchronous Server Gateway Interface) y wsgi(Web Server Gateway Interface):

Ambos son archivos para la configuración del proyecto y que trabaje en un servidor de producción de manera correcta. Puede consumir recursos a través de servicios web.

settings y urls: Nos permite configurar nuestro proyecto, como el lenguaje, zona horaria y motor de base de datos. Urls es un diccionario que va a contener el índice de nuestro proyecto, todas las urls que va a contener cada respuesta.

## Entrando a settings.py

### Patrón de arquitectura MTB (Model Template View)

Las plantillas (templates) son importantes en este modelo.



```

TEMPLATES = [ #Plantillas
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [], #Lista donde se encuentran las plantillas
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

```

## Bases de datos

Django tiene soporte para bases de datos SQLite y PostgreSQL.

```

# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases

DATABASES = { #Bases de datos
    'default': {
        'ENGINE': 'django.db.backends.sqlite3', #SQLite
        'NAME': 'djangoUniversity.db',
    }
}

```

## Lenguaje del proyecto y zona horaria

```

# Internationalization
# https://docs.djangoproject.com/en/3.2/topics/i18n/

LANGUAGE_CODE = 'es-mx'

TIME_ZONE = 'GMT-5'

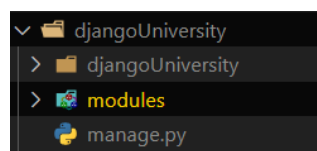
USE_I18N = True

USE_L10N = True

USE_TZ = False #Podemos no usar la zona horaria

```

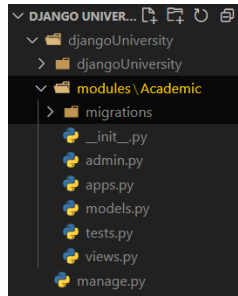
Creamos una carpeta de módulos



Dentro de esta carpeta ejecutamos el siguiente comando para iniciar una aplicación, creando una carpeta.

```
$ django-admin startapp Academic .
```

En esta carpeta/aplicación encontramos lo siguiente:



Models y views son elementos del patron de arquitectura MTV.

Admin nos sirve para registrar nuestras entidades en el panel de administración.

La carpeta migrations almacenará nuestras migraciones, este es un proceso de reflejar lo que en código se indica para que se refleje en la base de datos correspondiente.

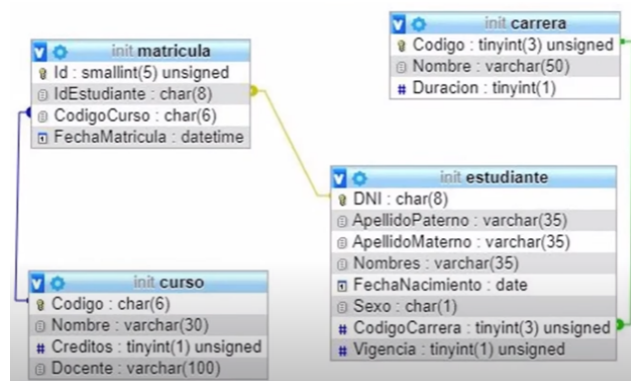
Comenzamos a trabajar en models.py: Los modelos se reflejan en nuestra base de datos como tablas.

Django posee un sistema ORM

**ORM (Object-Relational Mapping):**  
Mapeo objeto-relacional

Es una técnica de programación para convertir datos entre el sistema de tipos de datos usado en un lenguaje de programación que soporte el paradigma de la Programación Orientada a Objetos y los tipos de datos soportados por un motor de base de datos relacional específico, como medio de persistencia de datos.

Volviendo al diagrama entidad-relación vemos que tenemos 4 tablas, es decir 4 modelos a crear, vemos las relaciones de dependencia entre ellas.



Vemos que estudiante(tabla fuerte) depende de carrera(tabla débil), curso es una tabla fuerte y matrícula es una tabla débil que depende de cursos y estudiante.

Vamos a ir creando los modelos en el orden: carrera, estudiante, curso y al último matrícula.

Debemos registrar nuestra aplicación en apps.py de la siguiente manera

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'modules.Academic' #Registramos la aplicación academic
]
```

## Comenzamos a crear los modelos para cada tabla

```
from django.db import models
from django.db.models.enums import Choices
from django.db.models.expressions import F
#from django.db.models.fields import DurationField

# Create your models here.
#Creamos la clase carrera
class Carrer(models.Model):
    carrer_code=models.CharField(max_length=3,primary_key=True)
    nane=models.CharField(max_length=50)
    duration=models.PositiveSmallIntegerField(default=5)

class Student(models.Model):
    dni=models.CharField(max_length=8,primary_key=True)
    lastname1=models.CharField(max_length=35)
    lastname2=models.CharField(max_length=35)
    name=models.CharField(max_length=35)
    birthday=models.DateField()
    sex=[
        ('F', 'Female')
        ('M', 'Male')
    ]
    sexo=models.CharField(max_length=1,choices=sex,default='F')
    career=models.ForeignKey(Carrer,null=False,blank=False,on_delete=models.CASCADE)
    vigency=models.BooleanField(default=True)

    def fullName(self):
        txt="{0}, {1},{2}"
        return txt.format(self.lastname1,self.lastname2,self.name)

class Course(models.Model):
    code=models.CharField(max_length=6,primary_key=True)
    nombre=models.CharField(max_length=30)
    credits=models.PositiveSmallIntegerField()
    teacher=models.CharField(max_length=100)

class identificationNumber(models.Model):
    id=models.AutoField(primary_key=True)#Se crea un id uno a uno, campo incremental numérico
    student=models.ForeignKey(Student,null=False,blank=False,on_delete=models.CASCADE)#Relación de llave foranea
    course=models.ForeignKey(Course,null=False,blank=False,on_delete=models.CASCADE)
    dateIN=models.DateTimeField(auto_now_add=True)
```

Ahora en admin.py y registramos nuestros modelos para poderlos administrarlos en el panel de administración.

```

from djangoUniversity.modules.Academic.models import Carrer, Course, Student, identificationNumber
from django.contrib import admin
from modules.Academic.models import *

# Register your models here.

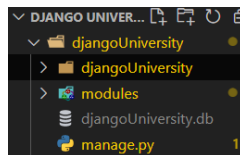
admin.site.register(Carrer)
admin.site.register(Student)
admin.site.register(Course)
admin.site.register(identificationNumber)

```

Regresamos al directorio de la carpeta del proyecto en la terminal, vamos a trabajar con el archivo `manage.py` ejecutamos el siguiente comando para hacer nuestra primera migración de nuestra base de datos.

```
$ python manage.py migrate
```

Vemos que se crea un archivo `.db` en formato SQLite



Ahora ejecutamos el siguiente comando para realizar la migración y que nuestros modelos se conviertan en tablas parte de la base de datos.

```

$ python manage.py makemigrations #esta linea no me funcionó
$ ./manage.py makemigrations Academic #Esta si!

```

Lo siguiente que haremos será registrar un super usuario que acceda al sistema y lo administre:

```
$ python manage.py createsuperuser
```

```

Nombre de usuario (leave blank to use 'usuario'): joelfestevez
Dirección de correo electrónico: joelito.402@gmail.com
Password:
Password (again):
Superuser created successfully.

```

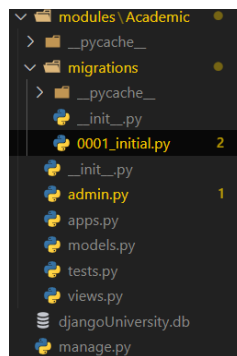
Cada que hagamos un cambio en la DB utilicemos los comandos `makemigrations` y `migrate` como buena práctica.

Es importante dar de alta la aplicación en los settings del proyecto

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'modules.Academic' #Registramos la aplicación Academic
]
```

Finalmente se crea la migración inicial al realizar una migración y aquí encontramos los módulos que creamos



Es importante que al realizar cambios en los modelos realicemos las migraciones correspondientes ejecutando los comandos de migración en el siguiente orden:

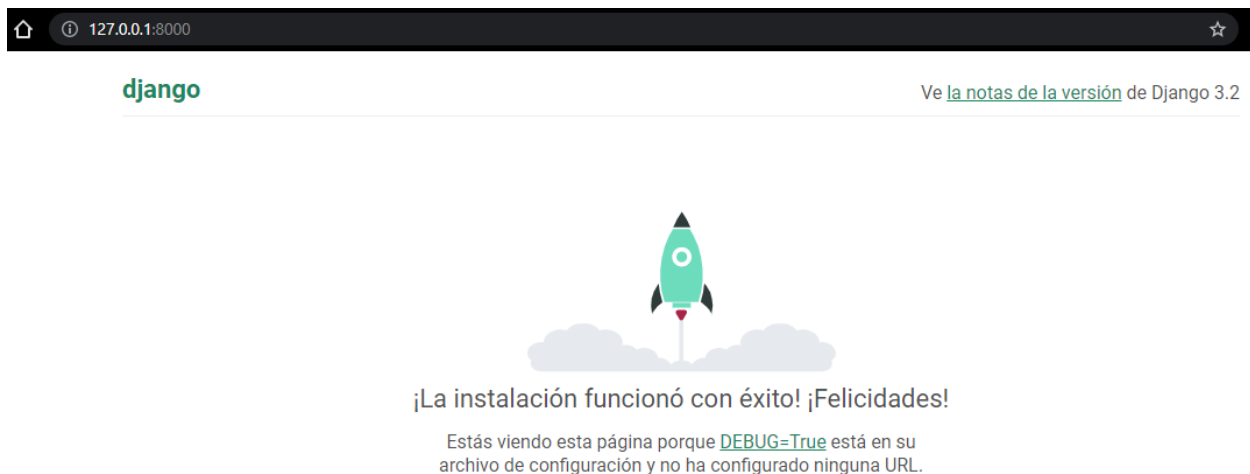
```
$ py manage.py migrate
$ py manage.py makemigrations
$ py manage.py migrate
```

De este modo nos aseguramos que todos los cambios que hagamos se verán reflejados en la base de datos.

## Inicializamos el servidor

```
$ py manage.py runserver
```

La aplicación se va a desplegar en el localhost

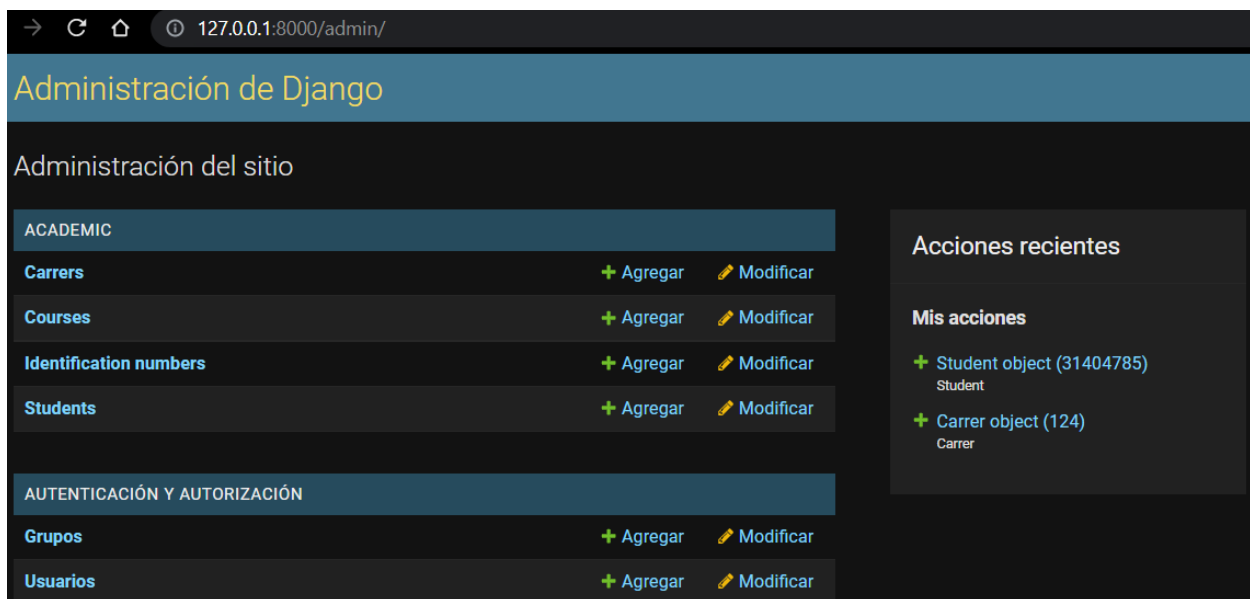


Para entrar al admin entramos con 127.0.0.1:8000/admin

Entramos con las credenciales con las que creamos nuestro superuser

Vamos a ver nuestro panel de administración, en donde podemos agregar carreras y usuarios.

En este caso me agregué a mi y añadí mi carrera con clave 124-Ingeniería Mecatrónica.



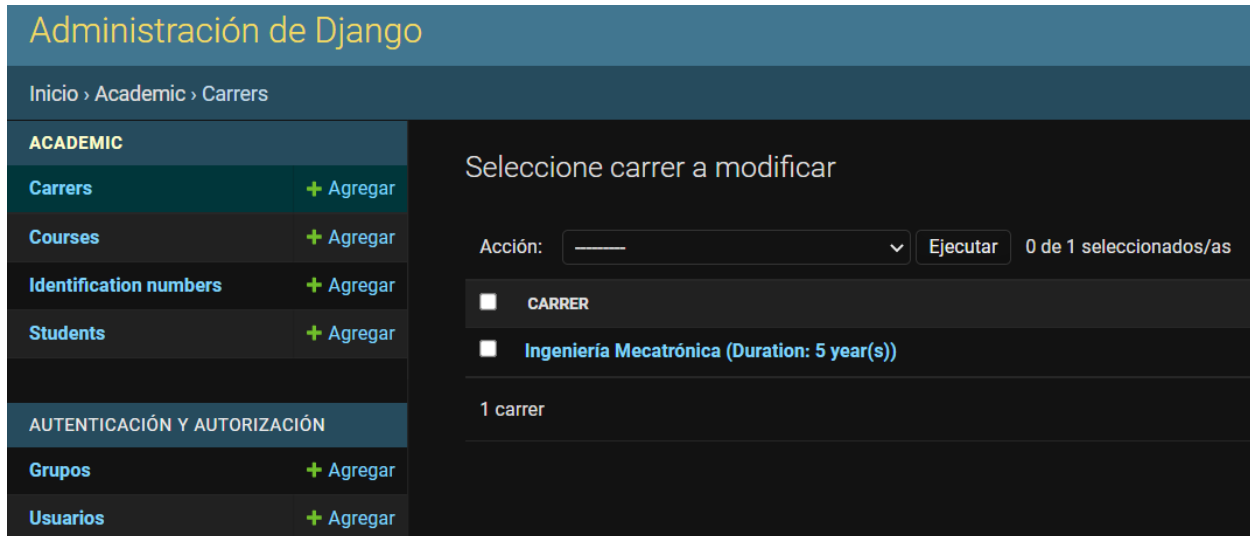
Vamos a mejorar la visualización, para que aparezcan los nombres tanto de la carrera como el de el estudiante.

Para el caso de carrera, en `models.py` vamos a crear una función debajo de `Carrer` que nos retorne un formato en string

```
def __str__(self) -> str:
    txt="{0} (Duration: {1} year(s))"
    return txt.format(self.name, self.duration)
```



Para detener el servidor presionamos en terminal ctrl+C  
 Volvemos a hacer las migraciones y volvemos a correr el servidor.  
 Vemos en admin en el apartado carrers que la función que hicimos es correcta.



Haremos lo mismo para el caso de los estudiantes, curso y matricula.

```
#Debajo de Student
def __str__(self) -> str:
    txt="{0} Career: {1}/{2}"
    if self.vigency:
        studentState="Vigent"
    else:
        studentState="Out"
    return txt.format(self.fullName(),self.career,studentState)

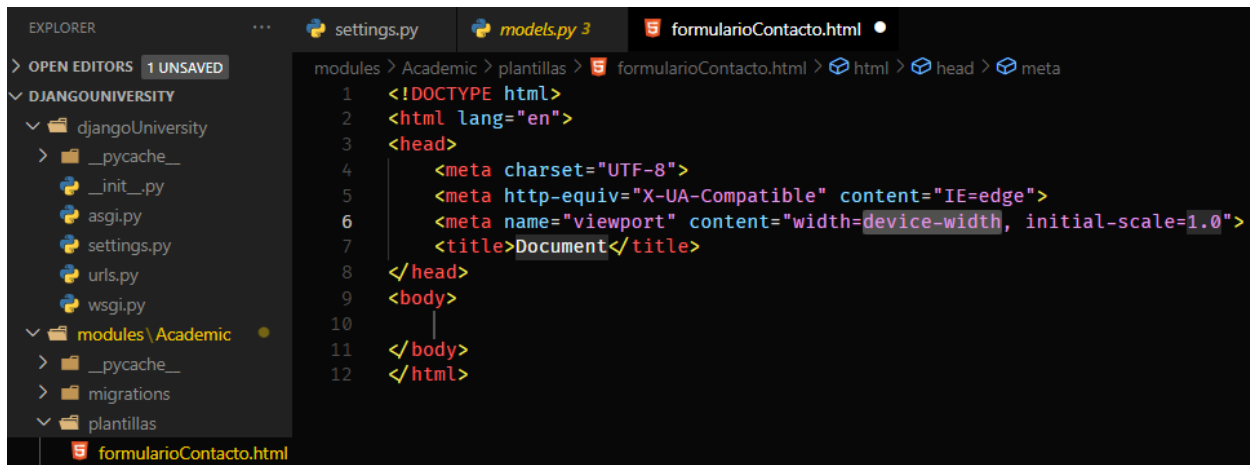
#Debajo de Course
def __str__(self) -> str:
    txt="{0} ({1} / Teacher: {2})"
    return txt.format(self.name,self.code,self.teacher)

#Debajo de identificationNumber
def __str__(self) -> str:
    txt="{0} matriculad{1} en el curso {2} / Date: {3}"
    if self.student.sex=="F": #Este if nos deja cambiar "o" y "a" si es hombre o mujer
        sexLetter="a"
    else:
        sexLetter="o"

    matriculeDate=self.dateIN.strftime("%A %d/%m/%Y %H:%M:%S")#Este es un formato para fecha
    return txt.format(self.student.fullName(), sexLetter, self.course, matriculeDate)
```

## Creación de Formulario para Enviar Correos

Detenemos el servidor y creamos un nuevo folder llamado Plantillas dentro de Académica.  
 Creamos un archivo formularioContacto.html



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Document</title>
8 </head>
9 <body>
10
11 </body>
12 </html>
```

Para crear la estructura básica de HTML 5 dentro de nuestro documento tecleamos html, sobre cargamos y aplicamos "html 5".

Comenzamos a dar estructura a nuestro formulario.

Revelación: WOW, que facil es escribir HTML con VSC. Maldito CCPM enseñando con el blog de notas.



```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>Formulario de Contacto</title>
8   </head>
9   <body>
10     <h1>Contáctenos</h1>
11     <form action="">
12       <p>Asunto: <input type="text" name="txtAsunto"></p>
13       <p>Email: <input type="email" name="txtEmail"></p>
14       <p>
15         Mensaje=<textarea name="txtMensaje" cols="30" rows="5" style="resize: none;"></textarea>
16       </p>
17       <input type="submit" value="Enviar">
18     </form>
19   </body>
20 </html>
```

Lo siguiente es crear una URL donde accedamos al formulario. Primero creamos una vista que nos redirija al formulario y luego configuramos esa vista para que forme parte de una ERL.

Vamos a Views.py dentro de Academic y definimos una primera vista, request es la petición y el render es la respuesta que se va a obtener a partir de la petición.

```
settings.py  formularioContacto.html  views.py 1
modules > Academic > views.py > formularioContacto
1  from django.shortcuts import render
2
3  # Create your views here.
4
5  def formularioContacto(request):
6      return render(request, "formularioContacto.html")
```

Para poder encontrar el formulario lo indicamos dentro de las plantillas. Vamos a Settings y en TEMPLATES, DIRS damos de alta la ruta donde están las plantillas.

Nota: Recuerda en tu caso cambiar las diagonales invertidas por diagonales normales

```
55  TEMPLATES = [ #Plantillas
56      {
57          'BACKEND': 'django.template.backends.django.DjangoTemplates',
58          'DIRS': ['C:/Users/Usuario/Desktop/Platzi/Proyectos/Django university/djangoUniversity/modules/Academic/plantillas'],
59          'APP_DIRS': True,
60          'OPTIONS': {
61              'context_processors': [
62                  'django.template.context_processors.debug',
63                  'django.template.context_processors.request',
64                  'django.contrib.auth.context_processors.auth',
65                  'django.contrib.messages.context_processors.messages',
66              ],
67          },
68      ],
69  ]
```

Después de esto vamos al archivo de URLS.py y configuramos la URL e importamos el fomulario desde modules/academica

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('fomularioContacto/', formularioContacto)
]
```

Terminado esto corremos el servidor nuevamente y entramos a la dirección pero en lugar de /admin entramos a /formularioContacto y tenemos el siguiente formulario básico:

← → ↻ 🏠 ⓘ 127.0.0.1:8000/formularioContacto/

## Contáctenos

Asunto:

Email:

Mensaje:

Dentro del formulario en HTML redirigimos a 'contactar' cuando el formulario se envíe y usamos el método POST. Luego utilizamos el paquete csrf token como método de seguridad para un splot malicioso que puede hacer que el sitio web permita comandos no autorizados.

```
<body>
  <h1>Contáctenos</h1>
  <form action="/contactar/" method="POST">{% csrf_token %}
```

El paquete django.contrib.csrf provee protección contra Cross-site request forgery (CSRF) (falsificación de peticiones inter-sitio).

Se presenta cuando un sitio Web malicioso induce a un usuario a cargar sin saberlo una URL desde un sitio al cual dicho usuario ya se ha autenticado, por lo tanto saca ventaja de su estado autenticado.

Configuramos la URL contactar. Vamos a crear una vista dentro de 'views.py' que nos permita crear esos parámetros.

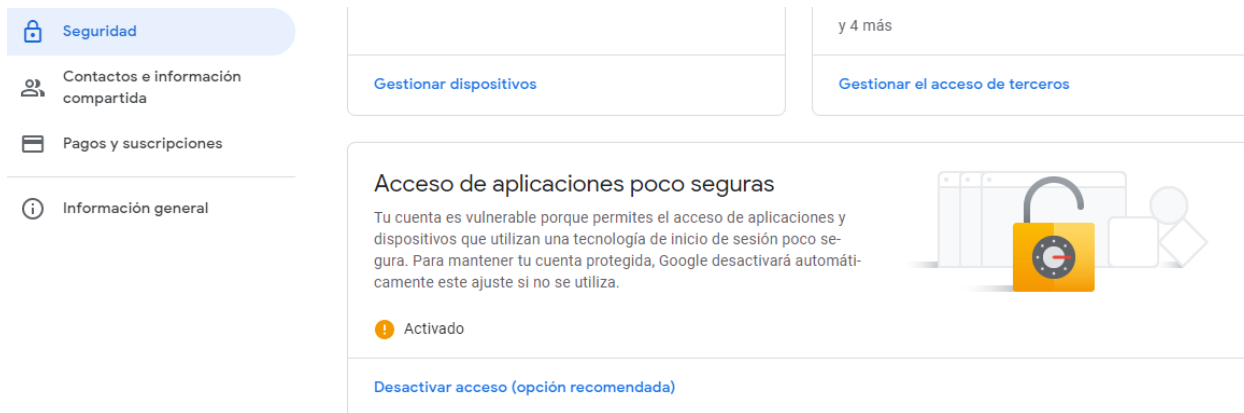
```
8 def contactar(request):
9     if request.method == "POST":
10         pass
```

Y dentro de los settings vamos a añadir el envío de correo electrónico como se muestra a continuación:

Nota: Obvio tienes que poner la contraseña de tu correo

```
123 STATIC_URL = '/static/'
124
125 EMAIL_BACKEND = "django.core.mail.backends.smtp.EmailBackend"
126 EMAIL_HOST="smtp.gmail.com"
127 EMAIL_USE_TLS = True
128 EMAIL_PORT=587
129 EMAIL_HOST_USER="joelito.402@gmail.com"
130 EMAIL_HOST_PASSWORD="*****"
```

Para poder hacer esto tenemos que ir a los ajustes del correo y activar en el apartado de seguridad lo siguiente:



Dentro de `views.py` vamos a definir el formulario contactar entonces si tenemos un método post entonces vamos a pedir cada una de las características del correo, en caso que sea get regresamos a formulario contacto.

`fail_silently` sirve para que se muestren errores si al enviar el correo algo sale mal.

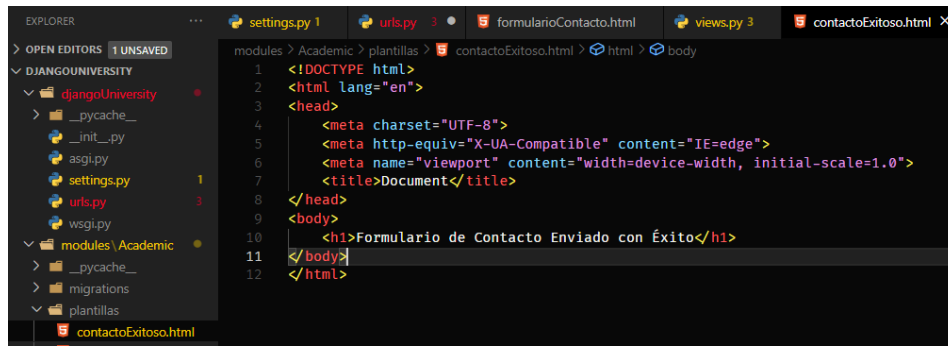
Se creó un nuevo html: `contactoExitoso.html` que se creará a continuación.

Recuerda importar la funcion `send_mail`:

```
settings.py 1 | urls.py 2 | formularioContacto.html | views.py 3 | contactoExitoso.html X
modules > Academic > views.py > ...
1 from django.shortcuts import render
2 from django.conf import settings
3 from django.core.mail import send_mail
```

```
1 from django.shortcuts import render
2 from django.conf import settings
3
4 # Create your views here.
5
6 def formularioContacto(request):
7     return render(request, "formularioContacto.html")
8
9 def contactar(request):
10     if request.method == "POST":
11         asunto=request.POST['txtAsunto']
12         mensaje=request.POST['txtMensaje']+" / Email: "+request.POST['txtEmail']
13         email_desde=settings.EMAIL_HOST_USER
14         email_para=('joelito.402@gmail.com')
15         send_mail(asunto, mensaje, email_desde, email_para, fail_silently=False) #Este fail sirve por si hay un error lo muestre
16         return render(request, 'contactoExitoso.html')
17     return render(request, 'formularioContacto.html') #En caso contrario "GET", retorno el formulario contacto
```

Creamos `contactoExitoso.html`, solo va a mostrar el mensaje de confirmación de enviado.



The screenshot shows a code editor with the Explorer sidebar on the left displaying the project structure. The main editor area shows the content of 'contactoExitoso.html', which is an HTML document with a title 'Document' and a body containing a heading 'Formulario de Contacto Enviado con Éxito'.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Document</title>
8 </head>
9 <body>
10   <h1>Formulario de Contacto Enviado con Éxito</h1>
11 </body>
12 </html>
```

Finalmente lo damos de alta en las URLs importante también contactar de views

```
16 from modules.Academic.views import formularioContacto
17 from django.contrib import admin
18 from django.urls import path
19 from modules.Academic.views import formularioContacto, contactar
20
21 urlpatterns = [
22     path('admin/', admin.site.urls),
23     path('formularioContacto/', formularioContacto),
24     path('contactar/', contactar)
25 ]
```

Volvemos a correr el servidor y la app debería funcionar correctamente...



!!!!ÉXITO!!!!!!