

Rayniel Ramos Gonzalez | Base de datos II

Manual Técnico

# Nombre del software:

GBDE (Gestor de Base de Datos de Estudiantes)

# Autor:

Rayniel Ramos González

# Estructura del proyecto:

# Carpeta TesisControlDB:

En esta carpeta se encuentran los distintos archivos donde se guardan las consultas realizadas de los distintos tipos de usuarios, cada archivo tiene el nombre del usuario que realizo la consulta, además está el manage.py, script que permite realizar todas las operaciones permitidas por Django sobre el proyecto, ver la documentación de Django 1.8 para más información. También se encuentra la base de datos, con el nombre db.sqlite3, al ser un archivo de base de datos sqlite3 acepta todas las opciones del manejo con la API de sqlite3, aunque no se aconseja agregar, modificar o eliminar datos, o realizar ninguna otra operación con ningún programa que no sea el script de manage.py.

# Carpeta .idea:

Información referente al IDE PyCharm.

# Carpeta .spyproject:

Información referente al IDE Spyder.

# Carpeta DB:

En esta carpeta se encuentra la app del proyecto como tal. En ella identificamos la carpeta migrations, la cual contiene todas las migraciones hechas a la base de datos del proyecto desde que comenzó a construirse. También está la carpeta template, en esta carpeta se encuentra las distintas páginas web (HTML) que no están incluidas en Django y son necesarias para realizar las consultas y mostrar distintos tipos de datos, el código de las mismas no debería ser de importancia al programador ya que hablamos de una suerte de interfaz visual.

El script admin dentro de esta carpeta contiene el código referente a las modificaciones hechas sobre el admin site de Django. En el mismo podemos ver:

# Register your models here.  
*from* django.contrib *import* admin  
  
*from* .models *import* Organismo  
*from* .models *import* Provincia  
*from* .models *import* Estudiante  
*from* .models *import* Curso  
*from* .models *import* Municipio  
*from* .models *import* Tesis  
*from* .models *import* PalabraClave  
  
*class* EstudianteAdmin(admin.ModelAdmin):  
 list\_display = ('CI', 'nombreApellidos', 'edad', 'email', 'sexo', 'cargo', 'centroTrabajo', 'reservaDeQueCuadro',  
 'provincia', 'organismo', 'municipio',)  
  
*class* OrganismoAdmin(admin.ModelAdmin):  
 list\_display = ('id', 'nombre', 'siglas',)  
  
*class* ProvinciaAdmin(admin.ModelAdmin):  
 list\_display = ('codigo', 'nombre',)  
  
*class* MunicipioAdmin(admin.ModelAdmin):  
 list\_display = ('id', 'nombre', 'provincia',)  
  
*class* PalabraClaveAdmin(admin.ModelAdmin):  
 list\_display = ('id', 'palabra',)  
  
*class* TesisAdmin(admin.ModelAdmin):  
 list\_display = ('id', 'titulo', 'ubicacion', 'organismo', 'link', 'Palabras', 'Estudiantes',)  
  
*class* CursoAdmin(admin.ModelAdmin):  
 list\_display = ('id', 'nombre', 'tipo', 'clasificacion', 'edicion',)  
  
admin.site.register(Organismo, OrganismoAdmin)  
admin.site.register(Provincia, ProvinciaAdmin)  
admin.site.register(Estudiante, EstudianteAdmin)  
admin.site.register(Curso, CursoAdmin)  
admin.site.register(Municipio, MunicipioAdmin)  
admin.site.register(Tesis, TesisAdmin)  
admin.site.register(PalabraClave, PalabraClaveAdmin)  
  
admin.site.site\_header = 'Base de datos'

Las clases herederas de ModelAdmin modifican la visualización del conjunto de datos de un tipo determinado, esto es, permite la visualización de los campos registrados en list\_display en las páginas correspondientes a mostrar el conjunto de datos de un tipo, posteriormente, el método admin.site.register() permite asignarle estas clases a los distintos modelos de la base de datos, es evidente que cada modelo tiene que tener el nombre del campo acorde al nombre que se encuentra en list\_display y se desea mostrar. Posteriormente se modifica el nombre de la cabecera de la página del sitio administración de la base de datos.

En el script url podemos ver:

*from* django.conf.urls *import* url  
  
*from* .views *import* ConStaticEsp  
*from* .views *import* ConStaticDip  
*from* .views *import* ConStaticView  
*from* .views *import* ConDinamicBusq  
*from* .views *import* ConStaticAutorNombre  
*from* .views *import* ConDinamicBusqReset  
  
# se pone el nombre de la app delante del nombre de la url para identificar que el nombre pertenece a la app, en  
# caso de existir otra template con el mismo nombre poder identificar esta que pertenece a la app DB  
urlpatterns = [  
 url(r'^conStaticDip/$', ConStaticDip, *name*='DB.ConStaticDip'),  
 url(r'^conStaticEsp/$', ConStaticEsp, *name*='DB.ConStaticEsp'),  
 url(r'^conStaticView/(?P<authorId>[0-9]+)/$', ConStaticView, *name*='DB.ConStaticView'),  
 url(r'^conDinamicBusq/$', ConDinamicBusq, *name*='DB.ConDinamicBusq'),  
 url(r'^conStaticAuthorName/(?P<courseId>[0-9]+)/$', ConStaticAutorNombre, *name*='DB.ConStaticAutorNombre'),  
 url(r'^conDinamicBusq/reset/$', ConDinamicBusqReset, *name*='DB.ConDinamicBusqReset'),  
]

El código anterior define las urls de la aplicación, les asocia una vista y un nombre para que puedan ser llamadas desde los templates de una forma más sencilla, tener presente que las url se almacenan en la lista con nombre urlpatterns para que pueda ser reconocida por Django.

En el script model.py se describe el modelo, se puede leer:

*from* django.db *import* models  
*from* django.core *import* validators  
*from* datetime *import* date  
  
# Create your models here  
*from* django.utils.html *import* format\_html  
  
  
*class* Organismo(models.Model):  
 nombre = models.CharField(*max\_length*=200, *verbose\_name*='Nombre\*')  
 siglas = models.CharField(*max\_length*=50, *verbose\_name*='Siglas', *blank*=*True*)  
  
 *def \_\_str\_\_*(*self*):  
 *return self*.nombre  
  
*class* Provincia(models.Model):  
 nombreProvincia = (('PinarDelRio', 'Pinar del Rio'),  
 ('Artemisa', 'Artemisa'),  
 ('LaHabana', 'La Habana'),  
 ('Mayabeque', 'Mayabeque'),  
 ('Matanzas', 'Matanzas'),  
 ('Cienfuegos', 'Cienfuegos'),  
 ('VillaClara', 'Villa Clara'),  
 ('SanctiSpiritus', 'Sancti Spiritus'),  
 ('CiegoDeAvila', 'Ciego de Avila'),  
 ('Camaguey', 'Camaguey'),  
 ('LasTunas', 'Las Tunas'),  
 ('Granma', 'Granma'),  
 ('Holguin', 'Holguin'),  
 ('SantiagoDeCuba', 'Santiago de Cuba'),  
 ('Guantanamo', 'Guantanamo'),  
 ('IslaDeLaJuventud', 'Isla de la Juventud'),)  
  
 codigo = models.IntegerField(*primary\_key*=*True*, *verbose\_name*='Codigo\*', *default*=1)  
 nombre = models.CharField(*max\_length*=25, *verbose\_name*='Nombre\*', *choices*=nombreProvincia, *default*='LaHabana',  
 *unique*=*True*)  
  
 *def \_\_str\_\_*(*self*):  
 *return self*.nombre  
  
*class* Estudiante(models.Model):  
 *class* Meta:  
 verbose\_name\_plural = 'Estudiantes'  
  
 sexoAutor = (('M', 'Masculino'), ('F', 'Femenino'))  
 maxEdadValidator = validators.MaxValueValidator(130, *message*='Aun se desconoce si alguien ha sobrepasado los 130')  
  
 CI = models.PositiveIntegerField(*primary\_key*=*True*, *verbose\_name*='Carnet de Identidad\*')  
  
 edad = models.PositiveSmallIntegerField(*verbose\_name*='Edad\*', *validators*=[maxEdadValidator])  
 nombreApellidos = models.CharField(*max\_length*=200, *verbose\_name*='Nombre y Apellidos\*')  
 email = models.EmailField(*verbose\_name*='Email', *blank*=*True*)  
 sexo = models.CharField(*verbose\_name*='Sexo\*', *max\_length*=5, *choices*=sexoAutor, *default*='F')  
 cargo = models.CharField(*verbose\_name*='Cargo\*', *max\_length*=100)  
 centroTrabajo = models.CharField(*max\_length*=200, *verbose\_name*='Centro de Trabajo\*')  
 reservaDeQueCuadro = models.CharField(*max\_length*=200, *verbose\_name*='Reserva de que Cuadro', *blank*=*True*)  
  
 organismo = models.ForeignKey(Organismo, *verbose\_name*='Organismo\*')  
 provincia = models.ForeignKey(Provincia, *verbose\_name*='Provincia\*')  
  
 municipio = models.ForeignKey('Municipio', *null*=*True*, *blank*=*True*)  
  
 *def \_\_str\_\_*(*self*):  
 *return self*.nombreApellidos  
  
*class* Curso(models.Model):  
 tipoCurso = (('Especialidad', 'Especialidad'), ('Diplomado', 'Diplomado'),)  
 clasificacionCurso = (('Administracion Publica', 'Administracion Publica'), ('Empresarial', 'Empresarial'),)  
  
 tipo = models.CharField(*max\_length*=25, *verbose\_name*='Tipo\*', *choices*=tipoCurso, *default*='Especialidad')  
 clasificacion = models.CharField(*max\_length*=25, *verbose\_name*='Clasificacion\*', *choices*=clasificacionCurso,  
 *default*='Empresarial')  
  
 edicion = models.PositiveIntegerField(*verbose\_name*='Edicion\*', *default*=1)  
 nombre = models.CharField(*max\_length*=100, *verbose\_name*='Nombre\*')  
  
   
  
 estudiantes = models.ManyToManyField(Estudiante, *verbose\_name*='Estudiantes\*')  
  
 *def \_\_str\_\_*(*self*):  
 *return self*.nombre  
  
*class* Municipio(models.Model):  
 nombreMunicipio = (  
 ('Pinar del Rio',  
 (  
 ('ConsolacionDelSur', 'Consolacion del Sur'),  
 ('Guane', 'Guane'),  
 ('LaPalma', 'La Palma'),  
 ('LosPalacios', 'Los Palacios'),  
 ('Mantua', 'Mantua'),  
 ('MinasDeMatahambre', 'Minas de Matahambre'),  
 ('PinarDelRio', 'Pinar del Rio'),  
 ('SanJuanYMartinez', 'San Juan y Martinez'),  
 ('SanLuis', 'San Luis'),  
 ('Sandino', 'Sandino'),  
 ('Vinales', 'Vinales'),  
 )  
 ),  
 ('Artemisa',  
 (  
 ('Alquizar', 'Alquizar'),  
 ('Artemisa', 'Artemisa'),  
 ('Bauta', 'Bauta'),  
 ('Caimito', 'Caimito'),  
 ('Guanajay', 'Guanajay'),  
 ('GuiraDeMelena', 'Guira de Melena'),  
 ('Mariel', 'Mariel'),  
 ('SanAntonioDeLosBanos', 'San Antonio de los Banos'),  
 ('Candelaria', 'Candelaria'),  
 ('SanCristobal', 'San Cristobal'),  
 ('BahiaHonda', 'Bahia Honda'),  
 )  
 ),  
 ('La Habana',  
 (  
 ('ArroyoNaranjo', 'Arroyo Naranjo'),  
 ('Boyeros', 'Boyeros'),  
 ('CentroHabana', 'Centro Habana'),  
 ('Cerro', 'Cerro'),  
 ('Cotorro', 'Cotorro'),  
 ('DiezDeOctubre', 'Diez de Octubre'),  
 ('Guanabacoa', 'Guanabacoa'),  
 ('LaHabanaDelEste', 'La Habana del Este'),  
 ('LaHabanaVieja', 'La Habana Vieja'),  
 ('LaLisa', 'La Lisa'),  
 ('Marianao', 'Marianao'),  
 ('Playa', 'Playa'),  
 ('PlazaDeLaRevolucion', 'Plaza de la Revolucion'),  
 ('Regla', 'Regla'),  
 ('SanMiguelDelPadron', 'San Miguel del Padron'),  
 )  
 ),  
 ('Mayabeque',  
 (  
 ('Batabano', 'Batabano'),  
 ('Bejucal', 'Bejucal'),  
 ('Guines', 'Guines'),  
 ('Jaruco', 'Jaruco'),  
 ('Madruga', 'Madruga'),  
 ('MelenaDelSur', 'Melena del Sur'),  
 ('NuevaPaz', 'Nueva Paz'),  
 ('Quivican', 'Quivican'),  
 ('SanJoseDeLasLajas', 'San Jose de las Lajas'),  
 ('SanNicolas', 'San Nicolas'),  
 ('SantaCruzDelNorte', 'Santa Cruz del Norte'),  
 )  
 ),  
 ('Matanzas',  
 (  
 ('Calimete', 'Calimete'),  
 ('Cardenas', 'Cardenas'),  
 ('CienagaDeZapata', 'Cienaga de Zapata'),  
 ('Colon', 'Colon'),  
 ('JagueyGrande', 'Jaguey Grande'),  
 ('Jovellanos', 'Jovellanos'),  
 ('Limonar', 'Limonar'),  
 ('LosArabos', 'Los Arabos'),  
 ('Marti', 'Marti'),  
 ('Matanzas', 'Matanzas'),  
 ('PedroBetancourt', 'Pedro Betancourt'),  
 ('Perico', 'Perico'),  
 ('UnionDeReyes', 'Union de Reyes'),  
 )  
 ),  
 ('Cienfuegos',  
 (  
 ('Abreus', 'Abreus'),  
 ('AguadaDePasajeros', 'Aguada de Pasajeros'),  
 ('Cienfuegos', 'Cienfuegos'),  
 ('Cruces', 'Cruces'),  
 ('Cumanayagua', 'Cumanayagua'),  
 ('Lajas', 'Lajas'),  
 ('Palmira', 'Palmira'),  
 ('Rodas', 'Rodas'),  
 )  
 ),  
 ('Villa Clara',  
 (  
 ('Caibarien', 'Caibarien'),  
 ('Camajuani', 'Camajuani'),  
 ('Cifuentes', 'Cifuentes'),  
 ('Corralillo', 'Corralillo'),  
 ('Encrucijada', 'Encrucijada'),  
 ('Manicaragua', 'Manicaragua'),  
 ('Placetas', 'Placetas'),  
 ('QuemadoDeGuines', 'Quemado de Guines'),  
 ('Ranchuelo', 'Ranchuelo'),  
 ('SanJuanDeLosRemedios', 'San Juan de los Remedios'),  
 ('SaguaLaGrande', 'Sagua la Grande'),  
 ('SantaClara', 'Santa Clara'),  
 ('SantoDomingo', 'Santo Domingo'),  
 )  
 ),  
 ('Sancti Spiritus',  
 (  
 ('Cabaiguan', 'Cabaiguan'),  
 ('Fomento', 'Fomento'),  
 ('Jatibonico', 'Jatibonico'),  
 ('LaSierpe', 'La Sierpe'),  
 ('SanctiSpiritus', 'Sancti Spiritus'),  
 ('Taguasco', 'Taguasco'),  
 ('Trinidad', 'Trinidad'),  
 ('Yaguajay', 'Yaguajay'),  
 )  
 ),  
 ('Ciego de Avila',  
 (  
 ('Baragua', 'Baragua'),  
 ('Bolivia', 'Bolivia'),  
 ('Chambas', 'Chambas'),  
 ('CiegoDeAvila', 'Ciego de Avila'),  
 ('CiroRedondo', 'Ciro Redondo'),  
 ('Florencia', 'Florencia'),  
 ('Majagua', 'Majagua'),  
 ('Moron', 'Moron'),  
 ('PrimerDeEnero', 'Primero de Enero'),  
 ('Venezuela', 'Venezuela'),  
 )  
 ),  
 ('Camaguey',  
 (  
 ('Camaguey', 'Camaguey'),  
 ('CarlosManuelDeCespedes', 'Carlos M. de Cespedes'),  
 ('Esmeralda', 'Esmeralda'),  
 ('Florida', 'Florida'),  
 ('Guaimaro', 'Guaimaro'),  
 ('Jimaguayu', 'Jimaguayu'),  
 ('Minas', 'Minas'),  
 ('Najasa', 'Najasa'),  
 ('Nuevitas', 'Nuevitas'),  
 ('SantaCruzDelSur', 'Santa Cruz del Sur'),  
 ('Sibanicu', 'Sibanicu'),  
 ('SierraDeCubitas', 'Sierra de Cubitas'),  
 ('Vertientes', 'Vertientes'),  
 )  
 ),  
 ('Las Tunas',  
 (  
 ('Amancio', 'Amancio'),  
 ('Colombia', 'Colombia'),  
 ('JesusMenendez', 'Jesius Menendez'),  
 ('Jobabo', 'Jobabo'),  
 ('Majibacoa', 'Majibacoa'),  
 ('Manati', 'Manati'),  
 ('PuertoPadre', 'Puerto Padre'),  
 ('LasTunas', 'Las Tunas'),  
 )  
 ),  
 ('Holguin',  
 (  
 ('Antilla', 'Antilla'),  
 ('Baguanos', 'Baguanos'),  
 ('Banes', 'Banes'),  
 ('Cacocum', 'Cacocum'),  
 ('CalixtoGarcia', 'Calixto Garcia'),  
 ('Cueto', 'Cueto'),  
 ('FrankPais', 'Frank Pais'),  
 ('Gibara', 'Gibara'),  
 ('Holguin', 'Holguin'),  
 ('Mayari', 'Mayari'),  
 ('Moa', 'Moa'),  
 ('RafaelFreyre', 'Rafael Freyre'),  
 ('SaguaDeTanamo', 'Sagua de Tanamo'),  
 ('Urbano Noris', 'Urbano Noris'),  
 )  
 ),  
 ('Granma',  
 (  
 ('BartolomeMaso', 'Bartolome Maso'),  
 ('Bayamo', 'Bayamo'),  
 ('BueyArriba', 'Buey Arriba'),  
 ('Campechuela', 'Campechuela'),  
 ('CautoCristo', 'Cauto Cristo'),  
 ('Guisa', 'Guisa'),  
 ('Jiguani', 'Jiguani'),  
 ('Manzanillo', 'Manzanillo'),  
 ('MediaLuna', 'Media Luna'),  
 ('Niquero', 'Niquero'),  
 ('Pilon', 'Pilon'),  
 ('RioCauto', 'Rio Cauto'),  
 ('Yara', 'Yara'),  
 )  
 ),  
 ('Santiago de Cuba',  
 (  
 ('Contramaestre', 'Contramaestre'),  
 ('Guama', 'Guama'),  
 ('Mella', 'Mella'),  
 ('PalmaSoriano', 'Palma Soriano'),  
 ('SanLuis', 'San Luis'),  
 ('SantiagoDeCuba', 'Santiago de Cuba'),  
 ('SegundoFrente', 'Segundo Frente'),  
 ('SongoLaMaya', 'Songo-La Maya'),  
 ('TercerFrente', 'Tercer Frente'),  
 )  
 ),  
 ('Guantanamo',  
 (  
 ('Baracoa', 'Baracoa'),  
 ('Caimanera', 'Caimanera'),  
 ('ElSalvador', 'El Salvador'),  
 ('Guantanamo', 'Guantanamo'),  
 ('Imias', 'Imias'),  
 ('Maisi', 'Maisi'),  
 ('ManuelTames', 'Manuel Tames'),  
 ('NicetoPerez', 'Niceto Perez'),  
 ('SanAntonioDelSur', 'San Antonio del Sur'),  
 ('Yateras', 'Yateras'),  
 )  
 ),  
 ('Isla de la Juventud',  
 (  
 ('IslaDeLaJuventud', 'Isla de la Juventud'),  
 )  
 ),)  
  
 *def \_\_str\_\_*(*self*):  
 *return self*.nombre  
  
 nombre = models.CharField(*max\_length*=25, *verbose\_name*='Nombre\*', *choices*=nombreMunicipio, *unique*=*True*)  
 provincia = models.ForeignKey(Provincia, *verbose\_name*='Provincia\*')  
  
*class* PalabraClave(models.Model):  
 *class* Meta:  
 verbose\_name\_plural = 'Palabras Claves'  
  
 palabra = models.CharField(*max\_length*=50, *verbose\_name*='Palabra\*', *unique*=*True*)  
  
 *def \_\_str\_\_*(*self*):  
 *return self*.palabra  
  
*class* Tesis(models.Model):  
 *class* Meta:  
 verbose\_name\_plural = 'Tesis'  
  
 link = models.URLField(*verbose\_name*='Enlace al articulo en Internet', *blank*=*True*)  
 titulo = models.CharField(*max\_length*=100, *verbose\_name*='Titulo\*')  
 tema = models.TextField(*verbose\_name*='Tema\*')  
 ubicacion = models.CharField(*max\_length*=100, *verbose\_name*='Ubicacion', *blank*=*True*)  
  
 organismo = models.ForeignKey(Organismo, *verbose\_name*='Organismo\*')  
 estudiantes = models.ManyToManyField(Estudiante, *verbose\_name*='Estudiantes\*')  
  
 palabras = models.ManyToManyField(PalabraClave, *verbose\_name*='Palabras claves\*')  
  
 *def \_\_str\_\_*(*self*):  
 *return self*.titulo  
  
 *def* Palabras(*self*):  
 html = '<select>'  
 *for* palabra *in self*.palabras.all():  
 html += '<option>' + palabra.\_\_str\_\_() + '</option>'  
  
 html += '</select>'  
  
 *return* format\_html(html)  
  
 *def* Estudiantes(*self*):  
 html = '<select>'  
 *for* estudiante *in self*.estudiantes.all():  
 html += '<option>' + estudiante.\_\_str\_\_() + '</option>'  
  
 html += '</select>'  
  
 *return* format\_html(html)

Cada clase define un nuevo modelo, las propiedades de la clase que son instancias de clases definidas en models, son o campos del modelo o relaciones entre los modelos. La clase Meta modifica información de metadatos acerca de un modelo, en este caso, modifica el nombre en plural del modelo, importante al mostrarlo en el sitio de administración. Los métodos (funciones) definidos en los modelos, no son más que ‘’llamables’’, esto es, métodos que pueden mostrar cierta información en el sitio de administración, en este caso estos métodos se solicitan para llamar en la propiedad list\_display vista anteriormente, retornan un fragmento HTML, que en este caso no es más que código HTML formateado en un string, el cual será embebido en los campos del modelo en el sitio de administración.

El script views.py es el que tiene las vistas, esto es, funciones que al pedir una url el navegador, son ejecutadas (cada url está asociada a una vista), realizando algún tipo de lógica y luego, generalmente, envían una página web al navegador. Las vistas son las siguientes:

*import* os  
*import* pickle  
*import* io  
*from* django *import* shortcuts  
*from* django.contrib.admin.sites *import* site  
*from* DB.code *import* Consul  
  
# Create your views here.  
  
*def* ConStaticDip(*request*):  
 *if* site.has\_permission(*request*):  
 query = Consul.DipConsul()  
 *return* shortcuts.render\_to\_response(os.path.join('DB', 'TemplateConStaticDiplomado.html'),  
 {'diplomados': query})  
 *raise* shortcuts.Http404()  
  
*def* ConStaticEsp(*request*):  
 *if* site.has\_permission(*request*):  
 query = Consul.EspConsul()  
 *return* shortcuts.render\_to\_response(os.path.join('DB', 'TemplateConStaticEspecialidad.html'),  
 {'especialidades': query})  
 *raise* shortcuts.Http404()  
  
*def* ConStaticView(*request*, *authorId*):  
 *if* site.has\_permission(*request*):  
 query = Consul.AuthorInfo(*authorId*)  
 *return* shortcuts.render\_to\_response(os.path.join('DB', 'TemplateConStaticView.html'), {'autor': query})  
 *raise* shortcuts.Http404()  
  
*def* ConStaticAutorNombre(*request*, *courseId*):  
 *if* site.has\_permission(*request*):  
 query = Consul.StaticViewConsul(*courseId*)  
 *return* shortcuts.render\_to\_response(os.path.join('DB', 'TemplateConStaticAutorNombre.html'), {'autores': query})  
 *raise* shortcuts.Http404()  
  
*def* ConDinamicBusq(*request*):  
 *if* site.has\_permission(*request*):  
  
 *if* 'raio' *in request*.GET:  
 # se le pregunta al request que usuario es y en dependencia se carga el pickle de ese usuario  
 radioId = *request*.GET['raio']  
 busq = *request*.GET['busqeda']  
  
 *if* busq == '':  
 *return* shortcuts.render\_to\_response(os.path.join('DB', 'TemplateConDinamicBuscador.html'))  
  
 *try*:  
 # si no existe se crea  
file = open(*request*.user.\_\_str\_\_(), 'rb') # se crea solo si se abre en modo de escritura  
 *except*:  
 # el file no se puede abrir por alguna razon se retorna una consulta vacia o que diga que hubo un error  
 # y que llame al administrador  
 print('error en abrir el file')  
 *pass  
  
 try*:  
 result = pickle.load(file)  
 print('se cargo la estructura')  
 *except*:  
 # se hace algo parecido a lo de arriba, si no se pudo cargar es porque el archivo aunque existe, esta  
 # vacio, en ese caso lo que se haria seria llenarlo con alguna especie de flag que me indique que debo  
 # realizar una consulta nueva y no cargar lo que esta y realizar una subconsulta  
 print('el file estaba vacio')  
 result = (*None*, 'new')  
  
  
 *try*:  
 file.close()  
 print('se cerro el file')  
 *except*:  
 *pass  
  
 if*(result[1] == 'new'):  
  
 file = open(*request*.user.\_\_str\_\_(), 'wb')  
  
 query = Consul.DinamicBusqConsul(radioId, busq)  
  
 estructura = (query, 'sub')  
  
 pickle.dump(estructura, file)  
  
 print('se realizo una consulta nueva, y se cargo la estructura al file')  
  
 *else*:  
 # result es el resultado de una subconsulta  
  
 file = open(*request*.user.\_\_str\_\_(), 'wb')  
  
 query = Consul.SubDinamicBusqConsul(radioId, busq, result[0])  
  
 estructura = (query, 'sub')  
  
 pickle.dump(estructura, file)  
  
 print('se realizo una subconsulta y se guardo la estructura')  
  
 *try*:  
 file.close()  
 print('se cerro el file')  
 *except*:  
 *pass  
  
 return* shortcuts.render\_to\_response(os.path.join('DB', 'TemplateConDinamicBuscador.html'),  
 {'resultado': query, 'count': query.count()})  
  
 *try*:  
 file = open(*request*.user.\_\_str\_\_(), 'rb')  
  
 estructura = pickle.load(file)  
  
 file.close()  
  
 *return* shortcuts.render\_to\_response(os.path.join('DB', 'TemplateConDinamicBuscador.html'),  
 {'resultado': estructura[0], 'count': estructura[0].count()})  
 *except*:  
 *return* shortcuts.render\_to\_response(os.path.join('DB', 'TemplateConDinamicBuscador.html'))  
  
 *raise* shortcuts.Http404()  
  
# ideal ver como poner el boton de restablecer en el mismo view del boton enviar consulta  
*def* ConDinamicBusqReset(*request*):  
 *if* site.has\_permission(*request*):  
  
 *try*:  
 print('se hizo reset')  
 file = open(*request*.user.\_\_str\_\_(), 'wb')  
 file.close()  
 *except*:  
 *pass  
  
 return* shortcuts.render\_to\_response(os.path.join('DB', 'TemplateConDinamicBuscador.html'))

La vista ConStaticDip es la encargada de realizar la consulta estática de los cursos por diplomados, la misma, al recibir una petición, verifica que el usuario que la realiza se encuentre registrado en el sitio y además haya iniciado sesión, de no ser así devuelve una página de error. Después de realizar la consulta, devuelve el template con los datos de la consulta.

La vista ConStaticEsp realiza lo mismo, pero ahora con una consulta de los cursos por especialidades.

La vista ConStaticAutorNombre es la encargada de, una vez seleccionado un curso, ya sea por especialidad o por diplomado en las vistas anteriores, desplegar una página intermedia con los nombres de los estudiantes que se encuentran en ese curso, posteriormente cada nombre es un enlace a los datos de ese estudiante.

La vista ConStaticView es la encargada de dado un autor devolver la página con información referente a él, se ejecuta al seguir el enlace del nombre de un estudiante que se encuentre en un curso, para esto realiza la consulta y devuelve la página con la información correspondiente.

La vista ConDinamicBusq es la más difícil de entender de todas, es la encargada de gestionar las consultas y subconsultas sobre las tesis, las cuales pueden ser filtradas por distintos parámetros. Para ello al igual que todas las vistas si un usuario ha iniciado sesión en el sitio, se verifica que se halla hecho un GET no solicitando la página, sino enviando información, los parámetros por los que se desean filtrar y el string que se debe hacer coincidir para buscar, si se envía esta información, entonces se comienza a verificar que se pueda abrir el archivo donde está la última consulta realizada por el usuario, de poderse abrir se carga la consulta y se realiza una subconsulta, de no poderse abrir, pues entonces se crea el archivo de nuevo y se realiza una consulta, posteriormente se guarda la consulta con un flag que indica que la próxima vez que se cargue el archivo debe realizarse una subconsulta sobre esos datos, en el caso de que el archivo no haya podido abrirse o no haya podido cargarse se realiza una consulta sobre la base de datos. Como las funciones open(), y dump() lanzan excepción en caso de no poder realizarse correctamente pues se han utilizado varios bloques de código try, except.

La vista ConDinamicBusqReset es la encargada de gestionar la acción del botón restablecer en la página de las búsquedas de las tesis, esta vista sencillamente borra el contenido del archivo del usuario que lo solicita y envía ‘’en blanco’’ la página de las búsquedas, de esta forma la vista anterior puede realizar una nueva consulta sobre la base de datos en vez de una subconsulta sobre los datos guardados en el archivo, puesto que se encuentra el archivo en blanco, esto es, no va a poder cargar los datos del archivo.

El script code.py contiene las funciones que son llamadas para las distintas consultas. Estas se encuentran siendo funciones (métodos) estáticos de la clase Consul.

La función DipConsul devuelve un objeto que contiene todos los cursos que son del tipo diplomado.

La función EspConsul devuelve un objeto que contiene todos los cursos que son del tipo de especialidad.

La función AuthorInfo dado el id de un estudiante devuelve el objeto que le representa y posee toda la información referente a ese estudiante.

La función StaticViewConsul dado el id de un curso devuelve un objeto que tiene todos los estudiantes en ese curso.

La función DinamicBusqConsul se encarga de dado el parámetro seleccionado al filtrar las tesis y el string que debe hacerse coincidir, realizar la consulta adecuada y devolver el objeto resultante de dicha consulta.

La función SubDinamicBusqConsul realiza algo similar a la anterior solo que recibe un objeto que ya es una consulta para realizar sobre el mismo una subconsulta.

La función FiltTesisPorNombreOrganismo realiza una consulta sobre las tesis por el organismo para el cual fueron hechas, o sea, dado el nombre de un organismo para el cual fueron hechas las tesis, buscar todas aquellas que fueron hechas para este organismo.

La función SubFiltTesisPorNombreOrganismo realiza algo similar pero sobre un objeto que ya contiene tesis de una consulta anterior, o sea, dado un objeto que contiene tesis de alguna consulta realizada, filtra las mismas por el nombre de un organismo que se le pasa como parámetro y devuelve estas nuevas tesis, subconjunto de las anteriores.

La función FiltTesisPorNombreAutor realiza una consulta sobre las tesis, buscando aquellas que contengan el nombre del estudiante pedido entre sus autores, y devuelve un objeto con las mismas.

La función SubFiltTesisPorNombreAutor realiza algo parecido a la anterior solo que esta vez como una subconsulta sobre un objeto que ya contiene tesis.

La función FiltTesisPorNombreProvincia realiza una consulta sobre las tesis, buscando aquellas tesis donde se encuentren estudiantes que sean de la provincia pedida.

La función SubFiltTesisPorNombreProvincia realiza algo similar a la función anterior, pero sobre una consulta de tesis que ya se ha realizado, o sea, dado ya un objeto que contiene tesis de una consulta ya realizada, las filtra siguiendo el criterio anterior.

La función FiltTesisPorCentroTrabajo realiza una consulta sobre las tesis buscando estudiantes que se encuentren entre sus autores y pertenezcan al centro de trabajo pedido.

La función SubFiltTesisPorCentroTrabajo realiza algo similar a la anterior, pero sobre un objeto que ya contiene tesis, o sea, realiza una subconsulta.

La función FiltTesisPorPalabras realiza una consulta sobre las tesis filtrando por las palabras claves.

La función SubFiltTesisPorPalabras realiza algo similar a la anterior solo que sobre un objeto que ya contiene tesis, realiza una subconsulta.

La función FiltTesisPorCargo realiza una consulta sobre las tesis buscando aquellas que tengan estudiantes en el cargo pedido.

La función SubFiltTesisPorCargo realiza algo similar a la anterior solo que sobre un objeto que ya contiene tesis de otra consulta realizada.

*from* .models *import* \*  
  
*class* Consul:  
 @staticmethod  
 *def* DipConsul():  
 *return* Curso.objects.filter(*tipo*='Diplomado')  
  
 @staticmethod  
 *def* AuthorInfo(*id*):  
 *return* Estudiante.objects.get(*pk*=*id*)  
  
 @staticmethod  
 *def* EspConsul():  
 *return* Curso.objects.filter(*tipo*='Especialidad')  
  
 @staticmethod  
 *def* StaticViewConsul(*id*):  
 '''  
 :param id: string con el valor del id del elemento de la tabla  
 :return: un objeto tipo QuerySet  
 '''  
 *return* Curso.objects.get(*pk*=*id*).estudiantes.all()  
  
 @staticmethod  
 *def* DinamicBusqConsul(*idradio*, *textbox*):  
 '''  
 Realiza una consulta basada en los parametros seleccionados en la app  
 :param idradio: tipo string, se pasa el value de un radio button de la app para saber la opcion  
 seleccionada  
 :param textbox: tipo string, se pasa el texto de un textbox con lo que se desea buscar  
 :return: un objeto de tipo QuerySet de Django, con los elementos de la consulta realizada  
 '''  
 *if idradio* == 'byOrganism':  
 *return* FiltTesisPorNombreOrganismo(*textbox*)  
  
 *elif idradio* == 'byProvince':  
 *return* FiltTesisPorNombreProvincia(*textbox*)  
  
 *elif idradio* == 'byWorkCenter':  
 *return* FiltTesisPorCentroTrabajo(*textbox*)  
  
 *elif idradio* == 'byCargo':  
 *return* FiltTesisPorCargo(*textbox*)  
  
 *elif idradio* == 'byWord':  
 *return* FiltTesisPorPalabras(*textbox*.strip().split(','))  
  
 *else*: # se filtro por autor por defecto  
 *return* FiltTesisPorNombreAutor(*textbox*)  
  
 @staticmethod  
 *def* SubDinamicBusqConsul(*idradio*, *textbox*, *query*):  
 *if idradio* == 'byOrganism':  
 *return* SubFiltTesisPorNombreOrganismo(*textbox*, *query*)  
  
 *elif idradio* == 'byProvince':  
 *return* SubFiltTesisPorNombreProvincia(*textbox*, *query*)  
  
 *elif idradio* == 'byWorkCenter':  
 *return* SubFiltTesisPorCentroTrabajo(*textbox*, *query*)  
  
 *elif idradio* == 'byCargo':  
 *return* SubFiltTesisPorCargo(*textbox*, *query*)  
  
 *elif idradio* == 'byWord':  
 *return* SubFiltTesisPorPalabras(*textbox*.strip().split(','), *query*)  
  
 *else*: # se filtro por autor por defecto  
 *return* SubFiltTesisPorNombreAutor(*textbox*, *query*)  
  
*def* FiltTesisPorNombreOrganismo(*organism*):  
 '''  
 Reliza la consulta de las tesis filtrando por el organismo, o sea, dado el organismo, devuelve las tesis  
 que fueron realizadas para el mismo  
 :param organism: un string, el organismo por el cual se va a filtrar, se usa contains  
 :return: el QuerySet de Django conteniendo las tesis que cumplen con la consulta pedida  
 '''  
 *return* Tesis.objects.filter(*organismo\_\_nombre\_\_icontains*=*organism*)  
  
*def* FiltTesisPorNombreAutor(*autor*):  
 '''  
 Realiza una consultas sobre las tesis buscando por autor, o sea, para aquellos autores que tengan nombre que  
 esten contenidos en el parametro, devuelve las tesis en la que se han visto involucrados  
 :param autor: string con el nombre del autor a buscar  
 :return: un objeto tipo QuerySet de Django con las tesis encontradas  
 '''  
 *return* Tesis.objects.filter(*estudiantes\_\_nombreApellidos\_\_icontains*=*autor*)  
  
*def* FiltTesisPorNombreProvincia(*prov*):  
 *return* Tesis.objects.filter(*estudiantes\_\_provincia\_\_nombre\_\_icontains*=*prov*)  
  
*def* FiltTesisPorCentroTrabajo(*centro*):  
 *return* Tesis.objects.filter(*estudiantes\_\_centroTrabajo\_\_icontains*=*centro*)  
  
*def* FiltTesisPorCargo(*cargo*):  
 *return* Tesis.objects.filter(*estudiantes\_\_cargo\_\_icontains*=*cargo*)  
  
*def* FiltTesisPorPalabras(*palabras*):  
  
 query = Tesis.objects.filter(*palabras\_\_palabra\_\_icontains*=*palabras*[0])  
  
 *for* indice *in* range(1, len(*palabras*)):  
 query = query.filter(*palabras\_\_palabra\_\_icontains*=*palabras*[indice])  
  
 *return* query  
  
*def* SubFiltTesisPorNombreOrganismo(*organism*, *query*):  
 *return query*.filter(*organismo\_\_nombre\_\_icontains*=*organism*).distinct()  
  
*def* SubFiltTesisPorNombreAutor(*autor*, *query*):  
 *return query*.filter(*estudiantes\_\_nombreApellidos\_\_icontains*=*autor*).distinct()  
  
*def* SubFiltTesisPorNombreProvincia(*prov*, *query*):  
 *return query*.filter(*estudiantes\_\_provincia\_\_nombre\_\_icontains*=*prov*).distinct()  
  
*def* SubFiltTesisPorCentroTrabajo(*centro*, *query*):  
 *return query*.filter(*estudiantes\_\_centroTrabajo\_\_icontains*=*centro*).distinct()  
  
*def* SubFiltTesisPorCargo(*cargo*, *query*):  
 *return query*.filter(*estudiantes\_\_cargo\_\_icontains*=*cargo*).distinct()  
  
*def* SubFiltTesisPorPalabras(*palabras*, *query*):  
  
 result = *query*.filter(*palabras\_\_palabra\_\_icontains*=*palabras*[0])  
  
 *for* indice *in* range(1, len(*palabras*)):  
 result = result.filter(*palabras\_\_palabra\_\_icontains*=*palabras*[indice])  
  
 *return* result.distinct()