PROCATE

MANUAL TÉCNICO

Contenido

Introducción

En el siguiente manual se presentan los aspectos que permitirán dar el mantenimiento preventivo y correctivo al sistema. Se detallara todos los aspectos importantes del programa, como las funciones más principales que se consideren, los dispositivos y toda librería que sea usada.

Dentro de nuestro manual, se encuentra una serie de funciones de código que le darán a conocer la forma lógica del funcionamiento del sistema que ayudara a su mejora y aprovechamiento del mismo, donde denotamos que se ha trabajado para lograr un sistema de calidad, ya que se ha puesto detalle en cada una de los requerimientos establecidos por el usuario, para que se pueda determinar que "ProCate" es un sistema apto para la utilización de la institución, dedicada a la impartición de clases de catequistas y que requiera un sistema que le ayude a la administración de la información de catequizandos de una manera óptima.

Cabe destacar que se ha realizado una auditoria para llevar un mejor control de las funciones con las que cuenta el sistema.

Objetivos

- Dar una breve descripción de las diferentes librerías que se utilizaron para la realización del sistema.
- Describir las diferentes funcionalidades que se utilizaron, de los cuales nos ofrece el framework Django.
- Dar a conocer las diferentes funcionalidades que presenta el sistema.
- Instruir al usuario final que pueda agregar una funcionalidad al sistema.

Librerías Utilizadas

Crispy-forms :

Esta herramienta nos proporciona un filtro |crispy y una etiqueta {% crispy %} que nos permitirá tomar el control sobre el comportamiento de las representaciones de formas de Django muy fácilmente. Además se integra muy bien en la paquetería de Django, por lo que no nos 'estorbará' en ningún momento durante el desarrollo del código.

Django-registration:

Nos proporciona vistas y mecanismos predefinidos para los casos de uso más habituales vinculados al registro de usuarios. Además, podemos encontrar varios templates ya creados compatibles con django-registration para las vistas que necesita.

Psycopg2:

Es el adaptador PostgreSQL más popular para el lenguaje de programación Python . En esencia, implementa completamente las especificaciones Python DB API 2.0. Varias extensiones permiten el acceso a muchas de las características ofrecidas por PostgreSQL.

Pytz:

Esta biblioteca permite cálculos de zona horaria precisos y multiplataforma usando Python 2.4 o superior. También resuelve el problema de los tiempos ambiguos al final del horario de verano, sobre el cual puede obtener más información en la Referencia de la Biblioteca de Python (datetime.tzinfo).

Virtualenv:

Permite crear entornos Python aislados. Con esto queremos decir, que podremos mantener una aplicación sin tenerla que actualizar, ya que cualquier cambio en sus librerías o dependencias podría comprometer el correcto funcionamiento de la misma.

Reporlab:

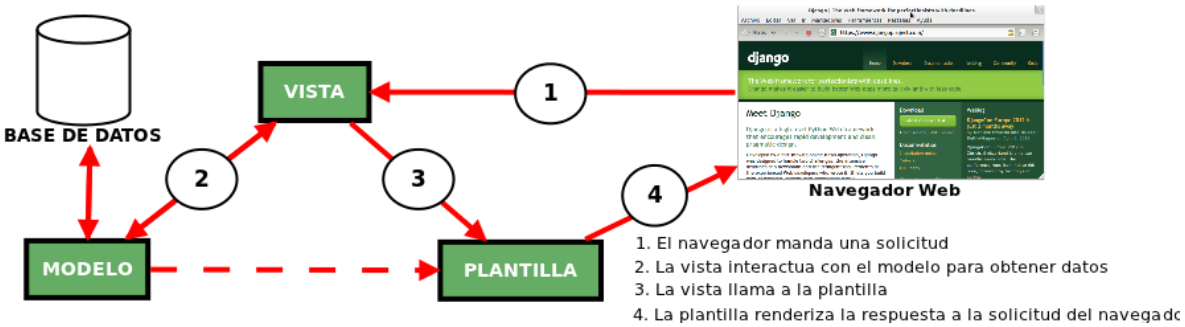
Es una librería que nos permite crear directamente documentos de en PDF (Formato de Documento Portable) usando Python como lenguaje de programación.

Patrón de trabajo MVT en Django

El patrón llamado MVT en **django** hace referencia a modelo vista template, y es sencillo de entender, el modelo sigue siendo el modelo, en este caso, la vista no es una vista, sino que más bien es un controlador que se llama vista, y el template son las vistas del MVC, es decir, los formularios van en template, los formularios hacen peticiones a las vistas', y las vistas obtienen datos de los modelos, es así de sencillo.

Para empezar a entender MVT debemos fijarnos en la analogía con MVC.

- El *modelo* en Django sigue siendo **modelo**
- La *vista* en Django se llama **Plantilla (Template)**
- El *controlador* en Django se llama **Vista**



Funcionamiento del MTV de Django

Algunos conceptos adicionales.

Modelo

El modelo define los datos almacenados, se encuentra en forma de clases de Python, cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros, posee métodos también. Todo esto permite indicar y controlar el comportamiento de los datos.

Vista

La vista se presenta en forma de funciones en Python, su propósito es determinar qué datos serán visualizad. El ORM de Django permite escribir código Python en lugar de SQL para hacer las consultas que necesita la vista. La vista también se encarga de tareas conocidas como el envío de correo electrónico, la autenticación con servicios externos y la validación de datos a través de formularios. Lo más importante a entender con respecto a la vista es que no tiene nada que ver con el estilo de presentación de los datos, sólo se encarga de los datos, la presentación es tarea de la plantilla.

Plantilla

La plantilla es básicamente una página HTML con algunas etiquetas extras propias de Django, en sí no solamente crea contenido en HTML (también XML, CSS, Javascript, CSV, etc).

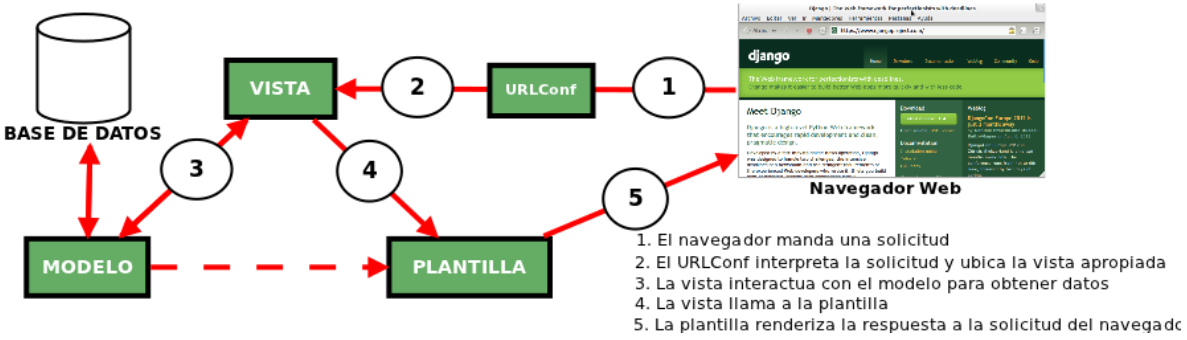
La plantilla recibe los datos de la vista y luego los organiza para la presentación al navegador web. Las etiquetas que Django usa para las plantillas permiten que sea flexible para los diseñadores del frontend, incluso tiene estructuras de datos como *if*, por si es necesaria una presentación lógica de los datos, estas estructuras son limitadas para evitar un desorden poniendo cualquier tipo de código Python.

Esto permite que la lógica del sistema siga permaneciendo en la vista.

La configuración de las rutas

Django posee un mapeo de URLs que permite controlar el despliegue de las vistas, esta configuración es conocida como URLConf. El trabajo del URLConf es leer la URL que el usuario solicitó, encontrar la vista apropiada para la solicitud y pasar cualquier variable que la vista necesite para completar su trabajo. El URLConf está construido con expresiones regulares en Python y sigue la filosofía de Python: Explicito es mejor que implícito. Este URLConf permite que las rutas que maneje Django sean agradables y entendibles para el usuario.

Si consideramos al URLConf en el esquema anterior tendríamos este resultado más completo.



Funcionamiento del MTV de Django y su URLConf

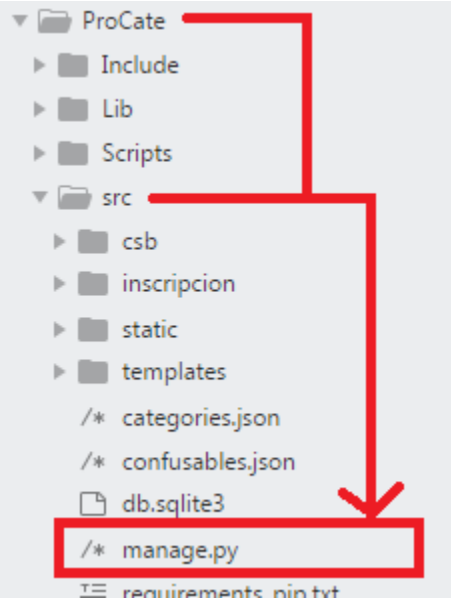
Los archivos predeterminados

Otra parte importante es entender el propósito de los archivos que se crean de manera predeterminada, estos son:

Archivos del proyecto

- **__init__.py**: Este es un archivo vacío que le dice a Python que debe considerar este directorio como un paquete de Python.

- **manage.py**: Este archivo contiene una porción de código que permite interactuar con el proyecto de Django de muchas formas. Si se desea mayor detalle se encuentra la documentación oficial con respecto a manage.py.



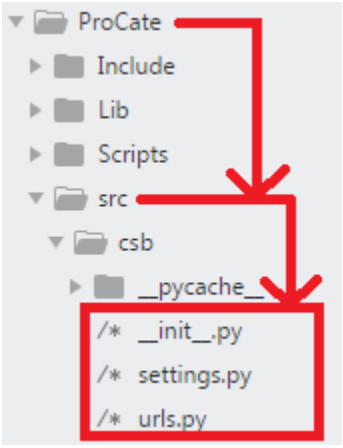
Directorio del archivo manage.py

- **settings.py**: Este archivo contiene todas las configuraciones para el proyecto, la documentación al respecto puede darnos más detalles de la configuración de un proyecto en Django.

- **urls.py**: Contiene las rutas que están disponibles en el proyecto, manejado por URLConf.

```
url(r'^comunidad/$', views.comunidad, name='comunidad'),
```

Este es un ejemplo de una URL que se utilizó en el proyecto, por lo que en el momento de entrar a la url de “comunidad/”, esta direccionara a la template que en la view este asignada.



Directorio de los archivos “_init_.py”, “settings.py” y “urls.py”

Archivos de la aplicación

- **__init__.py**: Este es un archivo vacío que le dice a Python que debe considerar este directorio como un paquete de Python.

- **models.py**: En este archivo se declaran las clases del modelo. Para la realización de una tabla es sencillo, tomaremos un ejemplo:

```
class Comunidad(models.Model):
    """docstring for Comunidad"""
    id_comunidad=models.AutoField(primary_key=True)
    nombre=models.CharField(max_length=100)

    def __unicode__(self): #Python 2
        return self.nombre

    def __str__(self): #Python 3
        return self.nombre
```

En este ejemplo vemos como se declara la tabla Comunidad con sus diferentes atributos (determinando el tipo de atributo, si es una llave primaria, además de predeterminar la longitud que este tendrá, entre otras cosas más que se le pueden agregar), y además con que atributo será reconocido por Python.

- **views.py**: En este archivo se declaran las funciones de la vista. Las funciones sirven para el manejo de la información, además de ser esta la que llama la URL, tomaremos la función de "comunidad" como ejemplo:

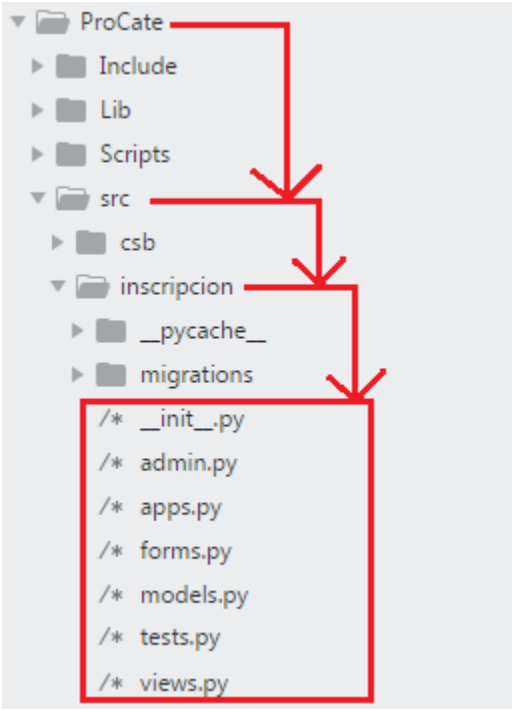
```
@login_required(login_url='inicio')
def comunidad(request):
    catequista=Catequista.objects.get(id_user=request.user.id)
    grupo=catequista.id_grupo

    if catequista.is_Admin:
        form=ComunidadForm(request.POST or None)
        objt=Comunidad.objects.all()
        if form.is_valid():
            form.save()
            return redirect('listaComunidad')
    else:
        return redirect('inicio')
    context={
        'comunidad':form,
        'coor':catequista,
        'grupo':grupo,

    }
    return render(request, "comunidad.html", context)
```

La función predetermina que para acceder a este tendrá que loguearse primero. Luego lo que se hace son consultas a la base de datos ya predeterminado por Python, en este caso se utiliza el id del usuario (línea 3) para obtener un objeto de catequista y luego obtener el grupo al que está asignado dicho catequista. Si este es Administrador, podrá utilizar la función en sí de guardar una nueva comunidad, si no es asi, se devolverá a la pantalla de inicio.

- **test.py**: En este archivo se declaran las pruebas necesarias para la aplicación.



Directorio de los archivos "_init_.py", "models.py", "views.py" y "test.py"

Agregar una Nueva Funcionalidad

Paso 1: Nos dirigimos al archivo “url.py” del proyecto y creamos una nueva url. Ejemplo:

```
url(r'^ejemplo/$', views.ejemplo, name='ejemplo'),
```

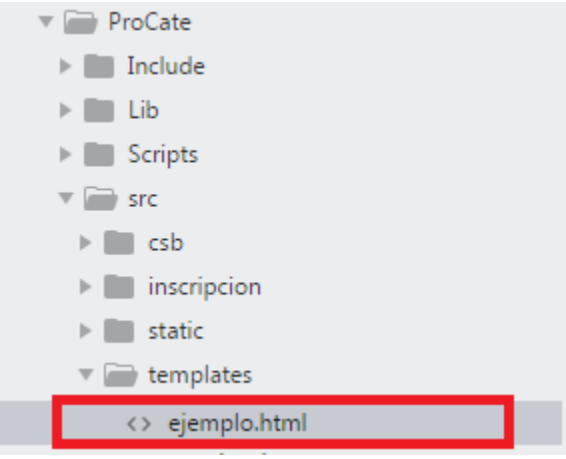
Al momento de ingresar a la url de “ejemplo/”, se irá directamente a la función “ejemplo” que se definirá en el archivo de view.

Paso 2: Al dirigirse al archivo “view.py” definimos la función. Ejemplo:

```
def ejemplo(request):
    return render(request,"ejemplo.html")
```

En este ejemplo al momento que sea llamada la función definida de “ejemplo”, nos redireccionara a la plantilla (template) con el nombre de “ejemplo.html”.

Paso 3: Creamos el archivo HTML del nombre del archivo al que se quiera dirigir la función.



Para el ejemplo que se ha tomado, el archivo HTML se dirigirá en el siguiente directorio. El ejemplo que se ha dado solo es una función que al ingresar a la dirección de la url “ejemplo/” este solo mostrara la plantilla del HTML “ejemplo.html”.

Funciones Principales de la Aplicación

Inscripción de Catequista

Objetivo: Hacer una nueva inscripción de Catequista asignándole un grupo en específico.

Paso 1: Nos dirigimos al archivo “views.py”.

Paso 2: Dentro del archivo se encuentra una función llamada “boleta”.

Paso 3: Dentro de esta función se encuentra diferentes consultas que se realizan a la base.

Paso 4: Esta función es llamada por el archivo “urls.py” del proyecto.

```
url(r'^inscribir/$', views.boleta, name='inscribir'),
```

Creación de una nueva Comunidad

Objetivo: Hacer un nuevo ingreso de una comunidad nueva.

Paso 1: Nos dirigimos al archivo “views.py”.

Paso 2: Dentro del archivo se encuentra una función llamada “comunidad”.

Paso 3: Dentro de esta función se encuentra diferentes consultas que se realizan a la base.

Paso 4: Esta función es llamada por el archivo “urls.py” del proyecto.

```
url(r'^comunidad/$', views.comunidad, name='comunidad'),
```

Imprimir Boleta

Objetivo: Hacer un nuevo ingreso de una comunidad nueva.

Paso 1: Nos dirigimos al archivo “views.py”.

Paso 2: Dentro del archivo se encuentra una función llamada “ImprimirBol”.

Paso 3: Dentro de esta función se encuentra diferentes consultas que se realizan a la base.

Paso 4: Esta función es llamada por el archivo “urls.py” del proyecto.

```
url(r'^ImprimirBol/(?P<id_cat>\d+)/$', views.ImprimirBol, name='imprimirbol'),
```

Mapeo de Entidades

El mapeo en Django es una técnica de programación para convertir datos entre un lenguaje de programación orientado a objetos y una base de datos relacional como motor de persistencia. El objetivo del ORM es el acceder a los modelos desde las vistas y desplegar resultados vía plantillas de HTML. Para ello se deben seguir unos pasos para las migraciones:

Se crea una migración para que el marco de migración de Django conozca los nuevos modelos que ha creado. Para hacerlo, ejecute el comando makemigrations:

```
python manage.py makemigrations
```

Se crea un archivo Python de migracion con los cambios del modelo

```
Migrations for 'blog':
  0001_initial.py:
    - Create model Category
    - Create model Post
```

Después de eso, debe aplicar la migración que acaba de crear y las migraciones ya existentes para crear el esquema de la base de datos utilizando el comando migrate:

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying sessions.0001_initial... OK
```

Definición de un modelo

Los modelos se guardan en el archivo “model.py” dentro de la carpeta “inscripción”.

Comunidad

```
class Comunidad(models.Model):
    """docstring for Comunidad"""
    id_comunidad=models.AutoField(primary_key=True)
    nombre=models.CharField(max_length=100)

    def __unicode__(self): #Python 2
        return self.nombre

    def __str__(self): #Python 3
        return self.nombre
```

Área

```
class Area(models.Model):
    """docstring for Area"""
    id_area=models.AutoField(primary_key=True)
    nombre=models.CharField(max_length=50)
    descripcion=models.CharField(max_length=256)

    def __unicode__(self): #pyton2
        return self.nombre

    def __str__(self): #pythin 3
        return self.nombre
```

Grupo

```
class Grupo(models.Model):
    """docstring for Grupo"""
    id_grupo=models.AutoField(primary_key=True)
    nombre=models.CharField(max_length=100)
    id_comunidad=models.ForeignKey(Comunidad)
    id_area=models.ForeignKey(Area)
    def __unicode__(self): #Python 2
        return self.nombre

    def __str__(self): #Python 3
        return self.nombre
```

Catequista

```
class Catequista(models.Model):
    """docstring for Catequista"""
    id_catequista=models.AutoField(primary_key=True)
    nombre=models.CharField(max_length=100, null=True)
    id_user=models.ForeignKey(User)
    id_comunidad=models.ForeignKey(Comunidad)
    id_area=models.ForeignKey(Area, null=True)
    id_grupo=models.ForeignKey(Grupo, null=True)
    is_Coordinador=models.BooleanField()
    is_Admin=models.BooleanField()
    is_Secretaria=models.BooleanField()

    def __str__(self):
        return '%s' %(self.nombre)
```

Catequizando

```
class Catequizando(models.Model):
    """docstring for Catequizando"""
    id_catequizando=models.AutoField(primary_key=True)
    id_grupo=models.ForeignKey(Grupo)
    nombre=models.CharField(max_length=100)
    nombre_madre=models.CharField(max_length=100, blank=True, null=True )
    nombre_padre=models.CharField(max_length=100, blank=True, null=True )
    email=models.EmailField(max_length=100)
    fecha_nacimiento=models.DateField(auto_now=False)
    lugar_de_bautizo=models.CharField(max_length=100, blank=True, null=True )
    fecha_de_bautizo=models.DateField(auto_now=False, null=True)
    is_activo=models.BooleanField()
    is_final=models.BooleanField()

    def __unicode__(self): #pyton2
        return self.nombre

    def __str__(self): #pythin 3
        return self.nombre
```

Asistencia

```
class Asistencia(models.Model):
    """docstring for Lista"""
    id_asistencia=models.AutoField(primary_key=True)
    id_grupo=models.ForeignKey(Grupo)
    id_catequizando=models.ForeignKey(Catequizando)
    fecha=models.DateField(auto_now_add=False,auto_now=False)

    def __unicode__(self):
        return str(self.fecha)

    def __str__(self):
        return str(self.fecha)
```

Boleta

```
class Boleta(models.Model):
    """docstring for Boleta"""
    id_boleta=models.AutoField(primary_key=True)
    id_catequizando=models.ForeignKey(Catequizando, related_name='+')
    id_catequista=models.ForeignKey(Catequista)
    id_area=models.ForeignKey(Area)
    id_comunidad=models.ForeignKey(Comunidad)
    fecha=models.DateTimeField(auto_now=True)
    fe_de_bautizo=models.BooleanField()
    partida_de_nacimiento=models.BooleanField()


    def __unicode__(self): #pyton2
        return str(self.id_boleta)

    def __tr__(self): #pythin 3
        return str(self.id_boleta)
```

Cambiar Validaciones

Existen varias formas de validar nuestra aplicación para cubrir aspectos de seguridad, algunas de ellas son:

- Validación de lado del cliente (Etiquetas HTML)
- Validación por medio de formularios.

Validación de lado del cliente:

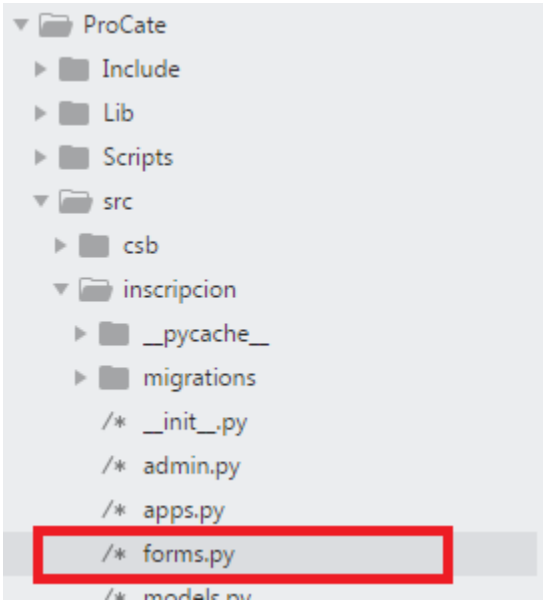
Podemos validar que los campos de un formulario sean requeridos al regresar el atributo *"required",* ejemplo:

```
<div class="col-md-9"> Nombre Catequizando:*<br>
  <input class="form-control" type="text" name="Nombre" required="true"></div>
```

El atributo *"required"* es un atributo booleano. Cuando está presente, este especifica que un campo debe ser rellenado antes de ser enviado el contenido del formulario.

Validación por medio de formularios:

Se ocupa otro archivo que está en la carpeta donde se encuentra nuestra aplicación, el nombre del archivo es *"forms.py"*.



En este archivo se hacen objetos de formularios prediseñados para solo llamarlos. Ejemplo de un formulario:

```
class ComunidadForm(forms.ModelForm):
    Nombre_comunidad=forms.CharField(required=True)
```

En este ejemplo se hace un formulario que contiene un input del tipo texto, ya que se llama al modelo de Comunidad, con el atributo nombre. Además cabe notar que el atributo tiene un *required* igualado a *True,* el cual hace el mismo funcionamiento que el *required* en el template.

Creación de Consultas

El framework de Django nos ayuda a que las consultas a nuestra base de datos sean de manera tan sencilla, esta herramienta es de mucha ayuda para la realización de funciones en el archivo de *"view.py".*

Un ejemplo tan sencillo es en el caso la obtención de todos las **Comunidades.**

```
comunidades=Comunidad.objects.all()
```

Este es en el caso de la obtención de todas las "comunidades". Para el caso de alguna comunidad en específico seria de esta manera:

```
comunidad=Comunidad.objects.get(id_comunidad=id_com)
```

Para este casi el id_comunidad es el id de la comunidad de la cual se quiere obtener, y el id_com es el valor de ese atributo.

Agregar nuevas URL al sistema

En este caso las URL como vimos anteriormente se agregan en el archivo del *"url.py"* del proyecto. Un ejemplo de ello puede ser:

```
url(r'^ejemplo/$', views.ejemplo, name='ejemplo'),
```

Por la cual "ejemplo/" es la dirección en el navegador que se tiene que escribir para acceder a ello, y además se le agrega la función del "view" a la cual tiene referencia.

Otra manera de la cual se puede escribir una URL es esta:

```
url(r'^ejemplo/(?P<id_ejemplo>\d+)$', views.ejemplo, name='ejemplo'),
```

En el cual se obtiene un parámetro y se manda a la función de "ejemplo" del atchivo "view.py", el nombre del parámetro en este ejemplo seria "id_ejemplo".

Control de Auditoria

Administrador	Nuevo	Editar	Eliminar	Buscar	Ver	Asignar	Generar PDF
Catequizandos							
Ingresar Catequizando	✓						
Ver Listado		✓			✓		
Grupos	✓	✓	✓		✓		
Gestión de Comunidades	✓	✓	✓		✓		
Gestión de Áreas	✓	✓	✓		✓		
Gestión de Usuarios							
Asignar Rol						✓	
Asignación de Área y Grupo						✓	
Activación de Usuario						✓	
Asistencias							
Pasar Asistencia	✓						
Historial					✓	✓	

Coordinador	Nuevo	Editar	Eliminar	Buscar	Ver	Asignar	Generar PDF
Catequizandos							
Ingresar Catequizando	✓						
Ver Listado		✓			✓		
Grupos	✓	✓	✓		✓		
Gestión de Áreas	✓	✓	✓		✓		
Gestión de Usuarios							
Asignación de Área y Grupo						✓	
Asistencias							
Pasar Asistencia	✓						
Historial					✓	✓	

Catequista	Nuevo	Editar	Eliminar	Buscar	Ver	Asignar	Generar PDF
Catequizandos							
Ingresar Catequizando	✓						
Ver Listado		✓			✓		
Asistencias							
Pasar Asistencia	✓						
Historial					✓	✓	

Catequista	Nuevo	Editar	Eliminar	Buscar	Ver	Asignar	Generar PDF
Imprimir Boleta				✓	✓		✓

Funciones Esenciales

Inscripción de Catequizando

Nombre Función: Boleta.

Descripción: Esta función se utiliza para hacer una nueva apertura de matrícula de un nuevo catequista. Se obtiene todos los datos de la persona, y se guarda en el modelo de la base de datos.

Ubicación: Procate/src/inscripción/views.py

Código:

```
def boleta(request):
            coor=Catequista.objects.get(id_user=request.user.id)
            ente=Catequista.objects.get(id_user=request.user.id)
            catequista=Catequista.objects.get(id_user=request.user.id)
            grupo=catequista.id_grupo
            print(ente.is_Secretaria)
            if coor.is_Secretaria:
                  redirect('inicio')
            else:
                  if request.user.is_active:
                        form=Inscribir(request.POST or None)

      catequista=Catequista.objects.get(id_user=request.user.id)
                        if coor.is_Admin:
                              form.fields['grupo'].queryset =
Grupo.objects.all()
                        else:
                              form.fields['grupo'].queryset =
Grupo.objects.filter(id_comunidad=catequista.id_comunidad)
                        titulo="Hola"
                        if request.user.is_authenticated():
                              titulo="Bienvenido %s" %(request.user)
                        if form.is_valid():
                              datos=form.cleaned_data
                              name=datos.get('Nombre')
                              name_mon=datos.get('Nombre_madre')
                              name_father=datos.get('Nombre_padre')
                              email=datos.get('Email')
                              date=datos.get('Fecha_de_nacimiento')
                              lugar=datos.get('Lugar_de_bautizo')
                              date_bau=datos.get('Fecha_de_bautizo')
                              fe=datos.get('fe_de_bautizo')
                              partida=datos.get('partida_de_nacimiento')
                              grupo=datos.get('grupo')
                              Catequizando.objects.create(id_grupo=grupo,
nombre=name, nombre_madre=name_mon, nombre_padre=name_father,
email=email, fecha_nacimiento=date, lugar_de_bautizo=lugar,
fecha_de_bautizo=date_bau ,is_activo=True, is_final=False)
```

```
      objt=Catequizando.objects.latest('id_catequizando')
                              ida=grupo.id_area
                              comunidad=grupo.id_comunidad
                              Boleta.objects.create(id_catequizando=objt,
id_area=ida,id_comunidad=comunidad,fe_de_bautizo=fe,
partida_de_nacimiento=partida, id_catequista=catequista )
            context={
                  "boletaForm":form,
                  'coor':ente,
                  'grupo':grupo,

            }
            return render(request, "inscribir.html", context)
```

Ingreso de Nueva Comunidad

Nombre Función: comunidad.

Descripción: Esta función se utiliza para el ingreso de una nueva comunidad.

Ubicación: Procate/src/inscripción/views.py

Código:

```
def comunidad(request):
      catequista=Catequista.objects.get(id_user=request.user.id)
      ente=Catequista.objects.get(id_user=request.user.id)
      grupo=catequista.id_grupo

      if catequista.is_Admin:
            form=ComunidadForm(request.POST or None)
            objt=Comunidad.objects.all()
            if form.is_valid():
                  form.save()
                  return redirect('listaComunidad')
      else:
            return redirect('inicio')
      context={
            'comunidad':form,
            'coor':ente,
            'grupo':grupo,

      }
      return render(request, "comunidad.html", context)
```

Pasar Asistencia

Nombre Función: asistencia.

Descripción: Esta función se utiliza para la realización de asistencia.

Ubicación: Procate/src/inscripción/views.py

Código:

```
def asistencia(request):
      form=AsistenciaForm(request.POST or None)
      catequista=Catequista.objects.get(id_user=request.user.id)
      lis=Catequizando.objects.filter(id_grupo=catequista.id_grupo).filte
r(is_activo=True)
      ente=Catequista.objects.get(id_user=request.user.id)
      grupo=catequista.id_grupo


      if request.method=='POST':
            if form.is_valid():
                  datos=form.cleaned_data
                  fecha=datos.get('fecha')
                  for element in request.POST.getlist('asis'):
                        grupo=grupo

      catequizando=Catequizando.objects.get(id_catequizando=element)
                        Asistencia.objects.create(id_grupo=grupo,
id_catequizando=catequizando, fecha=fecha)
      context={
            'asis':form,
            'cate':lis,
            'coor':ente,
            'grupo':grupo,
      }
      return render(request, "Asistencia.html", context)
```

Imprimir Boleta

Nombre Función: ImprimirBol.

Descripción: Esta función se utiliza para la impresión de boletas de los catequizandos.

Ubicación: Procate/src/inscripción/views.py

Código:

```
def ImprimirBol(request,id_cat):
      catequizando=Catequizando.objects.get(id_catequizando=id_cat)
      print(catequizando.nombre)
      response = HttpResponse(content_type='application/pdf')
      response['Content-Disposition']='attachment ; filename=Cerficado-
Confirma.pdf'
      buffer = BytesIO()
```

```
        c = canvas.Canvas(buffer, pagesize=A4)

        c.setFont('Helvetica',10)
        c.drawString(375,800,"Parroquia San Bartolomé Apósto")
        c.drawString(375,790,"San Bartolo, Ilopango")


        c.setFont('Helvetica',20)
        c.drawString(100,700,"CERTIFICADO DE PRIMERA COMUNION")

        c.setFont('Helvetica',13)
        c.drawString(50,620,"Por las presentes letras hago constar
que:     "+catequizando.nombre+"  ,")
        c.drawString(50,600,"bautizado el día 04/02/2014 en la
parroquia, después de haber sido")
        c.drawString(50,580,"debidamente catequizada en la doctrina de
Iniciación Cristiana, el sacramento de")
        c.drawString(50,560,"la Sagrada Comunión en esta Parroquia San
Bartolomé Apóstol de San Bartolo,")
        c.drawString(50,540,"Ilopango el día cuatro de diciembre del año
dos mil once.")

        c.drawString(50,480,"Y para los efectos que deseen y en derecho
corresponda, extiende la presente")
        c.drawString(50,460,"certificación en DIA ACTUAL.")
        c.line(210,210,400,210)
        c.setFont('Helvetica',15)
        c.drawString(200,180,"Pbro. Vinicio Alejandro Orizabal")
        c.drawString(230,150,"Párroco de Somoto")
        c.save()
        pdf=buffer.getvalue()
        buffer.close()
        response.write(pdf)

        return response
```