django-automatic-crud Documentation

Versión latest

11 de julio de 2022

Introduccion

CAPÍTULO 1

Django Automatic CRUD (CRUD Automáticos con Django)

Django Automatic CRUD es un proyecto que genera CRUDS automáticos para cada modelo que tenga la herencia indicada mas adelante. Estos CRUDS y URLS pueden ser de 2 tipos: **Normales y AJAX**.

1.1 Nota

**CRUDS Normales: ** Estos CRUDS son accesibles utilizando el Sistema de Plantillas de Django e incluyen validaciones de errores, existencia de templates, inicio de sesión y permisos.

**CRUDS AJAX: ** Estos CURDS son accesibles utilizando JavaScript o cualquier herramienta que permita realizar una petición a una URL indicada.

1.2 Características

- CRUDS automáticos con sólo crear los modelos.
- URLS generadas automáticamente para cada tipo de CRUD de modelo.
- Ruta para generación automática de un Reporte en formato Excel.
- Validación de Inicio de Sesión.
- Validación de Permisos.
- **CRUDS automáticos independientes, es decir, pueden generarse de los 2** tipos, sólo de uno o independiente.
- **Campos a excluir para listado, registro, edición y detalle de modelo** de forma dinámica.
- Mensajes de error automáticos y customizables.
- Nombre de templates para CRUDS customizables.
- Form de Django para CRUDS dinámico.

- Server-side.
- Paginación de datos.

1.3 Pre-Requisitos

- Django >= 2.2
- Python >= 3.3

1.4 Instalación Rápida

- Crea un entorno virtual e inicialo.
- Ejecuta el siguiente comando desde tu consola:

```
pip install django-automatic-crud
```

- Agrega automatic_crud a tu INSTALLED_APPS:

```
INSTALLED_APPS = [
    ...
    'automatic_crud',
    ...
]
```

1.5 Generación de CRUDS

- **Para cada modelo que desees generar los CRUDS, deben heredar de**
  BaseModel, por ejemplo:

```
from automatic_crud.models import BaseModel

class NewModel(BaseModel):
    ...
```

- Agrega la siguiente linea en tu archivo urls.py global:

```
path('automatic-crud/',include('automatic_crud.urls'))
```

- **Ahora, ingresa a tu navegador y escribe una ruta que no exista para**
  que Django pueda mostrarte todas las rutas existentes, te mostrará 14 rutas para cada modelo que herede de BaseModel, las cuales estarán dentro de la estructura de ruta: `http://localhost:8000/automatic-crud/` y tendrán el siguiente patrón:

```
automatic_crud/ app_name/ model_name / list / [name="app_name-model_name-list"]
automatic_crud/ app_name/ model_name / create / [name="app_name-model_name-create"]
automatic_crud/ app_name/ model_name / detail / <int:pk>/ [name="app_name-model_name-
↪detail"]
automatic_crud/ app_name/ model_name / update / <int:pk>/ [name="app_name-model_name-
↪update"]
```

(continué en la próxima página)

(proviene de la página anterior)

```
automatic_crud/ app_name/ model_name / logic-delete / <int:pk>/ [name="app_name-model_
↪name-logic-delete"]
automatic_crud/ app_name/ model_name / direct-delete / <int:pk>/ [name="app_name-model_
↪name-direct-delete"]
automatic_crud/ app_name/ model_name / excel-report / [name="app_name-model_name-excel-
↪report"]

automatic_crud/ ajax-app_name/ model_name / list / [name="app_name-model_name-list-ajax"]
automatic_crud/ ajax-app_name/ model_name / create / [name="app_name-model_name-create-
↪ajax"]
automatic_crud/ ajax-app_name/ model_name / detail / <int:pk>/ [name="app_name-model_
↪name-detail-ajax"]
automatic_crud/ ajax-app_name/ model_name / update / <int:pk>/ [name="app_name-model_
↪name-update-ajax"]
automatic_crud/ ajax-app_name/ model_name / logic-delete / <int:pk>/ [name="app_name-
↪model_name-logic-delete-ajax"]
automatic_crud/ ajax-app_name/ model_name / direct-delete / <int:pk>/ [name="app_name-
↪model_name-direct-delete-ajax"]
automatic_crud/ ajax-app_name/ model_name / excel-report / [name="app_name-model_name-
↪excel-report-ajax"]
```

Si quieres apoyar realizando una donación, puedes hacerla a este enlace:

- Donación al Proyecto

1.6 Redes Sociales

Web

Facebook

Instagram

Twitter

Youtube

Correo: developerpeperu@gmail.com

CAPÍTULO 2

BaseModel

BaseModel es el modelo padre de tipo **Abstracto**, tiene una herencia de `models.Model`, es decir:

```
from django.db import models

class BaseModel(models.Model):

    class Meta:
        abstract = True
```

Aquí es donde se han definido las características principales de Django Automatic CRUD, es el enlace con los modelos ya que, a través de la herencia, todo modelo obtendrá los siguientes campos:

```
 id = models.AutoField(primary_key = True)
 model_state = models.BooleanField(default = True)
 date_created = models.DateTimeField('Fecha de Creación', auto_now=False, auto_now_
↪add=True)
 date_modified = models.DateTimeField('Fecha de Modificación', auto_now=True, auto_now_
↪add=False)
 date_deleted = models.DateTimeField('Fecha de Eliminación', auto_now=True, auto_now_
↪add=False)

model_state es usado dentro de Django Automatic CRUD para la
 eliminación lógica.
```

Y los siguientes atributos:

```
all_cruds_types = True
normal_cruds = False
ajax_crud = False
server_side = False
exclude_model = False
login_required = False
```

(continué en la próxima página)

(proviene de la página anterior)

```
permission_required = ()
model_permissions = False
default_permissions = False
exclude_fields = ['date_created','date_modified','date_deleted','model_state']

success_create_message = "registrado correctamente!"
success_update_message = "actualizado correctamente!"
success_delete_message = "eliminado correctamente!"

error_create_message = "no se ha podido registrar!"
error_update_message = "no se ha podido actualizar!"
non_found_message = "No se ha encontrado un registro con estos datos!"

create_template = None
update_template = None
list_template = None
detail_template = None
```

2.1 Atributos de modelos que hereden de BaseModel

- **all_cruds_types** - si su valor es `True` generará CRUDS de tipo: Normales y AJAX, en `False` no generará ningún CRUD.
- **normal_cruds** - si *all*cruds_types_ es `True`, el valor de este campo no será tomado en cuenta, si *all*cruds_types_ es `False` y este campo es `True`, generará CRUDS de tipo `Normal`.
- **ajax_crud** - si *all*cruds_types_ es `True`, el valor de este campo no será tomado en cuenta, si *all*cruds_types_ es `False` y este campo es `True`, generará CRUDS de tipo `AJAX`.
- **server_side** - si su valor es `True` se realizará la paginación del lado del servidor, esto es válido sólo para *CRUDS de tipo AJAX*, retornará la siguiente estructura:

  ```
  {
      'length': # número de registros,
      'objects': # lista de datos por página
  }
  ```

- **exclude_model** - si su valor es `True`, no se generarán CRUDS para el modelo, aún cuando *all*cruds_types_ sea `True`.
- **login_required** - si su valor es `True`, solicitará que un quien realice la petición haya iniciado sesión. Se recomiendo realizar un `login(user)` de Django en la implementación de su sistema de Login.
- **permission_required** - tupla de permisos a solicitarse para un usuario que realice la petición a cualquier ruta de Django Automatic CRUD sólo si *model*permission_ es `True`.
- **model_permissions** - si su valor es `True`, solicitará permisos para el usuario que realice la petición.
- **default_permissions** - si su valor es `True`, los permisos a solicitar serán los básicos de Django, es decir, add,change,view,delete.
- **exclude_fields** - lista de campos excluidos, estos campos no serán tomados en cuenta para listar, editar, crear o cuando se obtenga el detalle de un registro. Por defecto los campos excluidos son los campos: `date_created, date_modified,date_deleted,model_state`.

- **success_create_message** - mensaje por defecto mostrado cuando se realiza un nuevo registro del modelo. Este campo es concatenado con el nombre del modelo, es decir: `{model.__name__} success_create_message`, por ejemplo: `Persona registrada correctamente`. **Válido sólo para CRUDS AJAX**.
- **success_update_message** - mensaje por defecto mostrado cuando se realiza una edición de un registro del modelo. Este campo es concatenado con el nombre del modelo, al igual que *success*create_message_. **Válido sólo para CRUDS AJAX**.
- **success_delete_message** - mensaje por defecto mostrado cuando se realiza una eliminación de un registro del modelo, ya sea eliminación lógica o directa. Este campo es concatenado con el nombre del modelo, al igual que *success*create_message_. **Válido sólo para CRUDS AJAX**.
- **error_create_message** - mensaje por defecto mostrado cuando ocurre un error al realizarse un nuevo registro del modelo. Este campo es concatenado con el nombre del modelo, al igual que *success*create_message_. **Válido sólo para CRUDS AJAX**.
- **error_update_message** - mensaje por defecto mostrado cuando ocurre un error al realizarse una edición de un registro del modelo. Este campo es concatenado con el nombre del modelo, al igual que *success*create_message_. **Válido sólo para CRUDS AJAX**.
- **non_found_message** - mensaje por defecto mostrado cuando no se encuentra un obtjeto solicitado. **Válido sólo para CRUDS AJAX**.
- **create_template** - nombre de template de creación para los CRUDS Normales del modelo. Por defecto el sistema solicita un template llamado `{model.__name__}_create.html`.
- **update_template** - nombre de template de edición para los CRUDS Normales del modelo. Por defecto el sistema solicita un template llamado `{model.__name__}_update.html`.
- **list_template** - nombre de template de listado para los CRUDS Normales del modelo. Por defecto el sistema solicita un template llamado `{model.__name__}_list.html`.
- **detail_template** - nombre de template de detalle para los CRUDS Normales del modelo. Por defecto el sistema solicita un template llamado `{model.__name__}_detail.html`.

NOTA

El nombre solicitado de forma automática por los templates para CRUDS Normales son generados por una función llamada build_template_name, puedes encontrar información en el apartado de **Funciones Extra - build_template_name**

CAPÍTULO 3

CRUDS Normales

3.1 BaseList

```
class BaseList(BaseCrudMixin,ListView):
    pass
```

Vista Basada en Clase encargada de generar y retornar el listado de registros para el modelo que se le haya indicado de forma automática.

Recibe herencia de BaseCrudMixin, la cuál se encarga de realizar las validaciones correspondientes a permisos y login_required, y de la clase Genérica de Django, ListView.

El listado de registros obtenidos para el modelo indicado serán retornados al template bajo el nombre de `object_list`

Si se coloca el parámetro `normal_pagination` en `True` se aplicará la paginación normal de Django, por defecto mostrará 10 elementos por página, sin embargo, esto se puede modificar con el atributo `values_for_page`

3.2 BaseCreate

```
class BaseCreate(BaseCrudMixin,CreateView):
    pass
```

Vista Basada en Clase encargada de generar y retornar el Form de Django para el agregar registros del modelo que se le haya indicado de forma automática.

Al registrar correctamente la instancia, redireccionará automáticamente a la ruta de Listado de CRUDS Normales.

Recibe herencia de BaseCrudMixin, la cuál se encarga de realizar las validaciones correspondientes a permisos y login_required, y de la clase Genérica de Django, CreateView.

Retorna el form de Django para el modelo al template bajo el nombre de `form`.

FORM PERSONALIZADO

Si se desea utilizar un Form de Django personalizado para el registro o edición de un modelo deberá sobreescribir los siguientes métodos en su modelo:

```
EJEMPLO

# form para crear
def get_create_form(self,form = None):
    from test_app.forms import CategoryForm
    self.create_form = CategoryForm
    return self.create_form

# form para actualizar
def get_update_form(self,form = None):
    from test_app.forms import CategoryForm
    self.update_form = CategoryForm
    return self.update_form
```

Siempre deberá importar el form personalizado **dentro de la función**, nunca fuera de ella, esto para evitar un error conocido como `Importación Circular`.

3.3 BaseDetail

```
class BaseDetail(BaseCrudMixin,DetailView):
    pass
```

Vista Basada en Clase encargada de retornar la instancias del modelo que se le haya indicado de forma automática.

Recibe herencia de BaseCrudMixin, la cuál se encarga de realizar las validaciones correspondientes a permisos y login_required, y de la clase Genérica de Django, DetailView.

Retorna la instancia del modelo al template bajo el nombre de `object`.

3.4 BaseUpdate

```
class BaseUpdate(BaseCrudMixin,UpdateView):
    pass
```

Vista Basada en Clase encargada de generar y retornar el Form de Django para la edición de una instancia del modelo que se le haya indicado de forma automática.

Al editar correctamente la instancia, redireccionará automáticamente a la ruta de Listado de CRUDS Normales.

Recibe herencia de BaseCrudMixin, la cuál se encarga de realizar las validaciones correspondientes a permisos y login_required, y de la clase Genérica de Django, UpdateView.

Retorna el form de Django para el modelo al template bajo el nombre de `form`.

Retorna la instancia del modelo al template bajo el nombre de `object`.

FORM PERSONALIZADO

Si se desea utilizar un Form de Django personalizado para el registro o edición de un modelo deberá sobreescribir los siguientes métodos en su modelo:

```
EJEMPLO

# form para crear
def get_create_form(self,form = None):
    from test_app.forms import CategoryForm
    self.create_form = CategoryForm
    return self.create_form

# form para actualizar
def get_update_form(self,form = None):
    from test_app.forms import CategoryForm
    self.update_form = CategoryForm
    return self.update_form
```

Siempre deberá importar el form personalizado **dentro de la función**, nunca fuera de ella, esto para evitar un error conocido como `Importación Circular`.

3.5 BaseDirectDelete

```
class BaseDirectDelete(BaseCrudMixin,DeleteView):
    pass
```

Vista Basada en Clase encargada de eliminar directamente de la Base de Datos la instancia del modelo que se le haya indicado de forma automática.

Recibe herencia de BaseCrudMixin, la cuál se encarga de realizar las validaciones correspondientes a permisos y login_required, y de la clase Genérica de Django, DeleteView.

Al eliminar correctamente la instancia, redireccionará automáticamente a la ruta de Listado de CRUDS Normales.

3.6 BaseLogicDelete

```
class BaseLogicDelete(BaseCrudMixin,DeleteView):
    pass
```

Vista Basada en Clase encargada de eliminar de forma lógica, es decir cambiando el campo `model_state` a `False` de la instancia del modelo que se le haya indicado de forma automática.

Recibe herencia de BaseCrudMixin, la cuál se encarga de realizar las validaciones correspondientes a permisos y login_required, y de la clase Genérica de Django, DeleteView.

Al eliminar correctamente la instancia, redireccionará automáticamente a la ruta de Listado de CRUDS Normales.

CAPÍTULO 4

CRUDS AJAX

4.1 BaseListAJAX

```
class BaseListAJAX(BaseCrud):
    pass
```

Vista Basada en Clase encargada de generar y retornar el listado de registros para el modelo que se le haya indicado de forma automática.

Recibe herencia de `BaseCrud`, la cuál se encarga de realizar las validaciones correspondientes a permisos y login_required.

El listado de registros obtenidos para el modelo será retornado en formato JSON y puede ser de 2 tipos:

SERVER SIDE

Retornará los datos paginados y sólo aquellos cuyo campo `model_state` sea `True` con la siguiente estructura por página:

```
{
    'length': # número de registros,
    'objects': # listado de registros
}

Ejemplo:

    {
        "length": 6,
        "objects": [
            {
                "pk": 1,
                "fields":{
                    "name": "abarrote",
```

(continué en la próxima página)

(proviene de la página anterior)

```
                    "modal_state": true,
                },
                "index": 1
            },
            {
                "pk": 1,
                "fields":{
                    "name": "carro",
                    "modal_state": true,
                },
                "index": 1
            }
        ]
    }
```

El campo `index` es la numeración para la tabla que se utilizará en caso desee enumerar cada item.

Y deben ser enviados en el request.GET los parámetros:

- **start** : número de elemento donde la página iniciará.
- **end** : número de elemento donde la página terminará.
- **order_by** : campo por el cuál los datos se ordenarán.

Por defectos estos valores serán 0, 10, id respectivamente.

Los campos que se hayan colocado como excluidos en el modelo, es decir en el campo `exclude_fields` del modelo no serán tomados en cuenta para el listado de datos

Para activar Server Side, revisar el apartado BaseModel

Para mas información o información aún mas detallada, revisar el siguiente vídeo Server Side con Django

NO SERVER SIDE

Retornará todos los registros del modelo que se encuentren en la Base de Datos cuyo campo `model_state` sea `True`.

```
Ejemplo:

    [
        {
            "pk": 1,
            "fields": {
                "model_state": true,
                "name": "abarrote"
            }
        },
        {
            "pk": 2,
            "fields": {
                "model_state": true,
                "name": "carro"
            }
        }
    ]
```

Para desactivar Server Side, revisar el apartado BaseModel

4.2 BaseCreateAJAX

```
class BaseCreateAJAX(BaseCrud):
    pass
```

Vista Basada en Clase encargada de realizar un nuevo registro para el modelo indicado automáticamente.

Recibe herencia de `BaseCrud`, la cuál se encarga de realizar las validaciones correspondientes a permisos y login_required.

Los nombres de los campos que deben ser enviados en la petición, request.POST, deben tener el mismo nombre que tienen estos en el modelo.

Al registrar correctamente la instancia o haber problemas al registrarla, retornará una respuesta de tipo JSON de la siguiente manera:

```
Registro Correcto

    {
        "message": "Categoria registrado correctamente!",
        "error": "Ninguno"
    }

Registro Incorrecto

    {
        "message": "Categoria no se ha podido registrar!",
        "error": {
                    "name": [
                        "This field is required."
                    ]
                }
    }
```

Los errores retornados son por campo y por defecto retornará los que Django haya reconocido automáticamente de los modelos, si desea utilizar errores personalizados deberá utilizar un Form de Django personalizado, el cual debe indicarlo en el modelo.

FORM PERSONALIZADO

Si se desea utilizar un Form de Django personalizado para el registro o edición de un modelo deberá sobreescribir los siguientes métodos en su modelo:

```
EJEMPLO

# form para crear
def get_create_form(self,form = None):
    from test_app.forms import CategoryForm
    self.create_form = CategoryForm
    return self.create_form

# form para actualizar
def get_update_form(self,form = None):
    from test_app.forms import CategoryForm
    self.update_form = CategoryForm
    return self.update_form
```

Siempre deberá importar el form personalizado **dentro de la función**, nunca fuera de ella, esto para evitar un error conocido como `Importación Circular`.

4.3 BaseDetailAJAX

```
class BaseDetailAJAX(BaseCrud):
    pass
```

Vista Basada en Clase encargada de retornar la instancia del modelo que se le haya indicado de forma automática.

Recibe herencia de `BaseCrud`, la cuál se encarga de realizar las validaciones correspondientes a permisos y login_required.

Retorna la información del objeto en formato JSON.

```
Ejemplo

    {
        "pk": 1,
        "fields": {
            "model_state": true,
            "name": "abarrote"
        }
    }
```

Los campos retornados son aquellos que no estén incluidos en el atributo del modelo `exclude_fields`

4.4 BaseUpdateAJAX

```
class BaseUpdateAJAX(BaseCrud):
    pass
```

Vista Basada en Clase encargada de realizar la actualización de un registro para el modelo indicado automáticamente.

Recibe herencia de `BaseCrud`, la cuál se encarga de realizar las validaciones correspondientes a permisos y login_required.

Los nombres de los campos que deben ser enviados en la petición, request.POST, deben tener el mismo nombre que tienen estos en el modelo.

Al actualizar correctamente la instancia o haber problemas al actualizar, retornará una respuesta de tipo JSON de la siguiente manera:

```
Actualización Correcto

    {
        "message": "Categoria actualizada correctamente!",
        "error": "Ninguno"
    }

Actualización Incorrecto
```

(continué en la próxima página)

(proviene de la página anterior)

```
    {
        "message": "Categoria no se ha podido actualizar!",
        "error": {
                    "name": [
                        "This field is required."
                    ]
                }
    }
```

Los errores retornados son por campo y por defecto retornará los que Django haya reconocido automáticamente de los modelos, si desea utilizar errores personalizados deberá utilizar un Form de Django personalizado, el cual debe indicarlo en el modelo.

FORM PERSONALIZADO

Si se desea utilizar un Form de Django personalizado para el registro o edición de un modelo deberá sobreescribir los siguientes métodos en su modelo:

```
EJEMPLO

# form para crear
def get_create_form(self,form = None):
    from test_app.forms import CategoryForm
    self.create_form = CategoryForm
    return self.create_form

# form para actualizar
def get_update_form(self,form = None):
    from test_app.forms import CategoryForm
    self.update_form = CategoryForm
    return self.update_form
```

Siempre deberá importar el form personalizado **dentro de la función**, nunca fuera de ella, esto para evitar un error conocido como `Importación Circular`.

4.5 BaseDirectDeleteAJAX

```
class BaseDirectDeleteAJAX(BaseCrud):
    pass
```

Vista Basada en Clase encargada de realizar la eliminación directa en la Base de Datos de un registro para el modelo indicado automáticamente.

Recibe herencia de `BaseCrud`, la cuál se encarga de realizar las validaciones correspondientes a permisos y login_required.

La respuesta dependerá de si se encontró o no el objeto que se desea eliminar.

```
Objeto encontrado

    {
        "message": "Categoria eliminado correctamente!",
```

(continué en la próxima página)

(proviene de la página anterior)

```
        "error": "Ninguno"
    }

Objeto no encontrado

    {
        "error": "No se ha encontrado un registro con estos datos."
    }
```

4.6 BaseLogicDeleteAJAX

```
class BaseLogicDeleteAJAX(BaseCrud):
    pass
```

Vista Basada en Clase encargada de realizar la eliminación lógica de un registro para el modelo indicado automáticamente, es decir, colocará el campo `model_state` en `False`.

Recibe herencia de `BaseCrud`, la cuál se encarga de realizar las validaciones correspondientes a permisos y login_required.

La respuesta dependerá de si se encontró o no el objeto que se desea eliminar.

```
Objeto encontrado

    {
        "message": "Categoria eliminado correctamente!",
        "error": "Ninguno"
    }

Objeto no encontrado

    {
        "error": "No se ha encontrado un registro con estos datos."
    }
```

CAPÍTULO 5

Reporte en Excel

Django Automatic CRUD incluye una ruta extra que genera un Reporte en formato de Excel, este reporte es automatizado, toma el modelo en cuestión y genera toda la estructura definida, es decir, `título, cabecera y cuerpo`.

El constructor de esta clase llamada `ExcelReportFormat`, define las siguientes variables y parámetros:

```
Parámetros:
    _app_name                   nombre de la aplicación donde está el modelo en cuestión.
    _model_name                 nombre del modelo a utilizar.

Variables:
    _app_name                   nombre de la aplicación donde está el modelo en cuestión.
    _model_name                 nombre del modelo a utilizar.
    __model                     modelo a usarse.
    __model_fields_names        lista de campos del modelo.
    __queryset                  queryset del modelo, contiene todos los registros de␣
↪este.
    __report_title              título del reporte.
    __workbook                  instancia de Workbook.
    __sheetwork                 hoja de Excel a utilizarse, por defecto es la primera.
```

La construcción del Reporte se realiza por etapas:

- **Etapa 1** - lo primero que se realiza es la asignación de las variables iniciales.

```
def __init__(self,__app_name:str,__model_name:str, *args, **kwargs):
    self.__app_name = __app_name
    self.__model_name = __model_name
    self.__model = get_model(self.__app_name,self.__model_name)
    self.__model_fields_names = get_model_fields_names(self.__model)
    self.__queryset = get_queryset(self.__model)
    self.__report_title = _excel_report_title(self.__model_name)
    self.__workbook = Workbook()
    self.__sheetwork = self.__workbook.active
```

- **Etapa 2** - se genera la cabecera del reporte, es decir, los nombres de cada campo como cabecera de las celdas donde se pintarán cada valor de cada campo, estos campos se obtienen de la variable `__model_fields_names`, en estos no se incluyen los campos que se encuentran en el atributo del modelo `exclude_fields`.

```
def __excel_report_header(self,row_dimension = 15, col_dimension = 25):
    pass
```

- **Etapa 3** - se pintan los valores para cada campos que se colocó en la cabecera de la tabla, dichos valores se obtienen de la variable `__queryset`.

```
def __print_values(self):
```

- **Etapa 4** - se construye la respuesta de tipo `ms-excel`, se toma el título que se genera con la función:

```
def _excel_report_title(__model_name: str):
    """
    Build report title with today date
    """


    date = datetime.now()
    title = "REPORTE DE {0} EN FORMATO EXCEL REALIZADO EN LA FECHA: {1}".format(
                                                                        __model_name.
↪upper(),
                                                                        "%s/%s/%s" %␣
↪(
                                                                            date.day,
↪date.month,
                                                                            date.year
                                                                        )
                                                                    )
    return title
```

Y se procede a construir una respuesta de tipo HttpResponse a la cual se le agrega el reporte.

```
def get_excel_report(self):
    pass
```

- Etapa 5 - finalmente hay una función que agrupa todos estos pasos, la cual construye como tal el reporte.

```
def build_report(self):
    """
    Build report call 2 functions: __excel_report_header and __print_values
    """

    self.__excel_report_header()
    self.__print_values()
```

Sin embargo, la clase `ExcelReportFormat` NO genera el retorno del Reporte en Excel que la URL dedicada llama, sino lo retorna la vista `GetExcelReport`, la cual recibe herencia de `BaseCrudMixin` y de `TemplateView`.

```
class GetExcelReport(BaseCrudMixin,TemplateView):
    pass
```

CAPÍTULO 6

Tipos de Datos

Los tipos de datos a los que hago referencia son a las Anotaciones de Python, una forma de realizar código Python estático, es sabido que Python es de tipado Dinámico, sin embargo, desde hace unas versiones podemos indicar tipado estático, claro está que no es tomado en cuenta por el intérprete sino sirve como ayuda para los programadores.

Los tipos de datos que encontrarás en Django Automatic CRUD son:

- **XLS** - Tipo de Dato de Excel.
- **Instance** - Tipo de Dato de Instancia.
- **URLList** - Tipo de Dato Listado de URLS.
- **DjangoForm** - Tipo de Dato Form de Django.
- **JsonResponse** - Tipo de Dato Respuesta en JSON.

CAPÍTULO 7

Resgistro de Modelos

La magia de la automatización de Django Automatic CRUD recae en esta funcionalidad, dentro de la instalación del paquete, es necesario incluir las rutas del paquete como tal en el archivo urls.py del proyecto donde se vaya a utilizar, esto se realiza por un motivo en específico que en si, es el motivo principal.

Cuando nosotros vinculamos estas rutas, lo que hacemos en si es llamar a la función `register_models` ya que el archivo urls de Django Automatic CRUD lo que contiene es:

```
from automatic_crud.register import register_models

urlpatterns = []

urlpatterns += register_models()
```

Esta función lo que realiza es una iteración de todos los modelos que existen dentro de las aplicaciones registradas en el proyecto donde se esté utilizando, excluyendo los modelos: `ContentType,LogEntry,Session,Permission, Group`.

Las validaciones que se hacen es que si o si el modelo debe ser de tipo `BaseModel` o que tenga los atributos de este tipo de modelos, se valida que el modelo tenga el atributo `exclude_model` en `True` y para agregar las URLS de cada tipo de CRUD que Django Automatic CRUD permite, es decir, tomando en cuenta los atributos del modelo `all_cruds_types, ajax_crud y normal_cruds`.

Finalmente se retornan las rutas generadas para cada modelo ya que en cada iteración por cada modelo se agregan las rutas a un listado de rutas que estarán en la variable `urlpatterns`.

CAPÍTULO 8

Funciones Extra

8.1 get_model

```
def get_model(__app_name:str,__model_name:str) -> Instance:
    # return the model corresponding to the application name and model name sent
    return apps.get_model(app_label = __app_name,model_name = __model_name)
```

- **__app_name** - Nombre de la aplicación donde está el modelo en cadena de texto
- **__model_name** - Nombre del modelo en cadena de texto

Retorna el modelo en cuestión para el nombre de la aplicación y modelo indicados.

8.2 get_object

```
def get_object(model: Instance,pk: int):
    # return the record for a pk sended
    instance = model.objects.filter(id = pk,model_state = True).first()
    if instance:
        return instance
    return None
```

- **model** - Modelo a realizar la consulta.
- **pk** - ID de registro a buscar.

Retorna la instancia del modelo perteneciente al pk enviado.

8.3 get_model_fields_names

```
def get_model_fields_names(__model: Instance) -> List:
    # return a list of field names from a model
    return [name for name,_ in models.fields_for_model(__model).items()]
```

- **__model** - Modelo del cual se desea obtener los nombres de sus campos.

Retorna una lista con los nombres de los campos del modelo enviado.

8.4 get_queryset

```
def get_queryset(__model:Instance) -> Dict:
    # returns all records in a dictionary for a model
    return __model.objects.all().values()
```

- **__model** - Modelo del cual se desea obtener los nombres de sus campos.

Retorna todos los datos registrados en la base de datos para el modelo indicado. Esta función se utiliza en el Reporte en Excel generado automáticamente.

8.5 get_form

```
def get_form(form: DjangoForm,model: Instance) -> DjangoForm:
    """
    Return a Django Form for a model, also a Django Form can be indicated
    by default the Django Form will exclude the 'state' field from the model

    """


    if form is not None:
        return models.modelform_factory(model = model,form = form)
    else:
        return models.modelform_factory(model = model,exclude = ('model_state',))
```

- **model** - Modelo en el cual se desea basar el Form de Django a crearse.
- **form** - Form de Django opcional a utilizarse en la creación de un Form de Django basado en modelo.

Retorna un Form de Django basado en el modelo indicado. Opcionalmente recibe el parámetro form, el cuál se utilizará para generarlo el nuevo Form en caso se desee utilizar uno personalizado. Para que el Form se genere automáticamente sin necesidad de enviarle el parámetro `form`, este debe ser enviado como None.

8.6 build_template_name

```
def build_template_name(template_name: str,model: Instance,action:str) -> str:
    """
    Build template name with app label from model, model name and action(list,create,
↪update,detail)

    """


    if template_name == None:
        template_name = '{0}/{1}_{2}.html'.format(
                                        model._meta.app_label,
                                        model._meta.object_name.lower(),
                                        action
                                    )
    return template_name
```

- **model** - Modelo del cuál se desea generar los nombres de templates solicitados en CRUDS Normales.
- **template_name** - Nombre del template a utilizarse en la vista de CRUDS Normales.