

DT-DATA

DATA LICENCIAS MANUAL TECNICO

Manual técnico de Data Licencias

Esta es una aplicación para hacer cálculos de proyectos de arquitectura y generar reportes de proyectos.

Características Generales de Data Licencias

Área	Tecnología o especificación	Observación
Backend	Python, Django	python-3.7.10
Frontend	HTML3, CSS5 y JavaScript	
Base de Datos	PostgreSQL	
Servicio Cloud	Digital Ocean	
Arquitectura	MTV	Arquitectura por defecto de Django

Usuarios de la Aplicación

- Propietario: quién llena las cuentas.
- Proyectista: se encarga de trámites de solicitudes, registro y verificación dentro de la aplicación.
- Delegados: usuario con acceso a múltiples vistas.
- Arquitectos.
- Temp: usuarios temporales con acceso a todo el sistema para las configuraciones iniciales de la aplicación.

Nota: Se esta usando servidor SINCRONO porque ASÍNCRONO no es necesario en CAP.

Carpeta cap/:

+ Dentro de la carpeta cap se encuentra el código "**setting.py**" que contiene la configuración de la aplicación y del entorno de producción, test y debug. Este script contiene la variable que la define llamada DEBUG que esta inicializada en "True", para luego tomar el DEBUG de la configuración del entorno

```
DEBUG=config('DEBUG', cast=bool)
```

```
DEBUG = True
HEROKU = True
DEBUG = config('DEBUG', cast=bool)
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': config('DB_NAME'),
        'USER': config('DB_USER'),
        'PASSWORD': config('DB_PASSWORD'),
        'HOST': config('DB_HOST'),
        'PORT': config('DB_PORT', cast=int),
    }
}
```

La configuración esta almacenada en el entorno.

en la **THIRD_PARTY_APPS** posee:

1. "ubigeos": librería que contiene serializadores que es útil para trabajar con los ubigeos.
2. "rest_framework": Se utilizara para la API.
3. "django_filters": Forma parte de las dependencias de "ubigeos".
4. "corsheaders": Habilitar el cors.

```
THIRD_PARTY_APPS = (  
    'ubigeos',  
    'rest_framework',  
    'django_filters',  
    'corsheaders',  
)
```

En el **MIDDLEWARE** posee:

"whitenoise": Para los estáticos y los estáticos están comprimidos para poder facilitar la rapidez de la descarga de la misma.

```
MIDDLEWARE = [  
    # 'django.template.context_processors.media',  
    'django.middleware.security.SecurityMiddleware',  
    'whitenoise.middleware.WhiteNoiseMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'corsheaders.middleware.CorsMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
)
```

En **TEMPLATES** posee:

```
apps.dashboard.context_processors.global_context
```

Solo trae los roles de usuario.

Digital Ocean en el servidor esta false y en settings en true para que así cuando Digital Ocean esta en true trae la base de datos del entorno la url que proporciona Digital Ocean. No se esta utilizando Time zone, igualmente esta configurado.

MEDIA_ROOT = BASE_DIR/ 'media' : Hace referencia al entorno pero como esta en producción ya no esta siendo utilizada.

MEDIA_URL = '/media/': si se esta utilizando.

LOGGING = Sirve para saber exactamente que cosas importantes aparecerán en el Log.


CORS_ALLOWED_ORIGINS = Esta la ip publica que se tendrá que agregar el dominio que le asigna el cap para poder permitir conexiones a través del cors headers.

Las url están en su propio archivo ".py" (login, logout, dashboard, admin, profiles, projects, evaluations, ubigeos)

EL **Email** ya esta configurado para mandar envíos por correos y ya previamente configurado en el entorno.


Space de digital option también configurado y funcional para trabajar los medias de cap.

hay un archivo .txt donde se dan recomendaciones para los deploy.

 Al momento de instalar la app solamente los staff pueden entrar a los require.

En setting.py además está el "**server.log**" que almacenas los eventos ocurridos en el servidor. Se presenta también un script un script llamado "**star.sh**" que inicia el servidor Linux donde va a correr la aplicación.

 **storage.py:** Esta siendo manejado por botoS3.

 **url.py:** Contiene las aplicaciones principales Login, logout, url de admin(django), profiles, projects, evaluations y ubigeos(se usa desde la API). También contiene la URL de la media. La redirección será a dashboard que seria como la pagina principal de CAP.

Carpeta dbs/:

```
last.backup  
restore.bat
```

Carpeta media/:

Media no se usa solo cuando el DEBUG esta en false o cuando no hay configuración de space.

Carpeta static/:

Componentes externos contiene la DEMO 1.0, contiene e css: todos los estilos de admin, app, auth, blue, error y master. Los scripts más importante son app.css y admin.css que poseen todo el estilo de la interfaz.

Carpeta dynamic-formset/:

Form set dinámicos: donde se generan los formularios dinámicos, además sirve para no tener que renderizarse o actualizar en el servidor sino por medio de JavaScript, utiliza jQuery.

Carpeta font/:

Letra de estilos personalizados por el cliente. Esta la carpeta img que contiene los avatars, logos y firma.

Carpeta js/:

Cada carpeta que contiene el js que contiene las librerías de js que serian el funcionamiento de la pagina en si.

Carpeta vendor/:

Contiene las librerías externas, Bootstrap, jQuery, fontawesome entre otros.

Carpeta templates/:

Esta organizada por aplicación (admin, base, dashboard, evaluations, pdf, profiles, projects y reports).

Carpeta base/:

contiene :

```
base.html
base_admins.html
base_pdf.html, base_public.html
base_reports.html.
```


Notas de despliegue e instalación

El proyecto contiene un archivo deploy.txt, donde recomendaciones para los deploy.

El url **dt-data-cap.Digital Oceanapp.com/set_initial_data/**: Al iniciar se deben establecer parámetros iniciales como tipos de Uso, subtipos de uso y tipos de obra, esta acción solo lo puede realizar el súper usuario.

"runtime.txt"

apps/:

 **Dashboard**: contiene las herramientas globales y administra todas las rutas que contiene en una sola url por lo tanto seria la aplicación principal. La lógica del dashboard es administrar todo lo del usuario, en /users esta el dashboard, en /delegates esta el dashboard de delegados, /technical-commission el dashboard de comisión técnica, de igual manera con /municipalities para los proyecto se usa la url/projects.

- Contiene las herramientas globales, atributos para los formularios. en el script **"attr.py"** hay funciones tales como el control de fecha y demás, atributos para formularios.
- El procesador de contexto(contiene los roles de usuario que serán utilizadas en las vistas). Script **"context_processors.py"** existen dichas funcionalidades de roles .
- Los emails, en el código **"email.py"** dentro dashboard esta el script que permite el envío de correo de la aplicación, ejemplos: el render de la vista, correo de validación de usuarios al momento de registro, correo de confirmación de registro en la aplicación mediante token , mensaje de bienvenida, reseteo de password, solicitud de ingreso, recibo de liquidación cuando caja aprueba un pago, notificación cuando se niega una solicitud.
- Contiene los roles de el propietario que esta habilitado para llenar encuestas, proyectista realiza la parte de registrar, solicitudes verificar ciclo de vida de las solicitudes y finalmente los delegados.
- También esta la confirmación de la cuenta, para resetear password también recibir la liquidación y también cuando se niega la solicitud o aprobada(se realiza captura de todo este tipo de notificaciones como constancia).
- Contiene los reportes en PDF. En el archivo **"pdf.py"**, se ejecutan los procesos de renderizado de vista y descargar de los pdf, tales como la liquidación mensual, reporte de delegados de sus evaluaciones, entro otros.
- El modelo municipalidad que tiene en común con las demás interfaces. Script **"models.py"**.
- Contiene su propio filtro (por ejemplo filtro de liquidación). Script **"filters.py"**.
- Contiene los reportes almacenados a través de json. Es un archivo que convierte de json a query, se emplea en varias partes de la aplicación. Script **"api.py"**.

- Contiene archivo **“choices.py”** para los elementos que no van a ser modificados como las ciudades, tipo de municipalidad, los meses, etc.
- El script **“charts.py”** donde se generan los charts de cada reporte.
- En el archivo de **“form.py”** están los métodos que generan los formularios de la aplicación.
- En **“tools.py”** las funciones a manera de herramientas, tales como la de rango de fechas, los help text, obtener el tiempo en string.

+ Evaluations: es una app que tiene dos funciones una son evaluaciones son evaluación de calidad y evaluación desempeño de delegados.

Estos scripts pueden ser modificado a través de el **“choice.py”**.

en **“models.py”** hay modelo de conocimientos técnicos, comisión técnica, calidad de servicio, desempeño del presidente, del primer delegado.

+ Profiles: Se trabaja los usuarios y sus perfiles, se manejan los perfiles de los proyectistas, propietarios, delegados, arquitectos, ingenieros. Esta aplicación permite mediante una realizar búsquedas mediante una api en el buscador (búsqueda rápida). También se puede buscar por nombre y por RUC.

- Contiene una API para la busca de arquitectos únicamente.
- Contiene una búsqueda rápida para todos los users.
- Contiene choices donde están todos los roles, tipo de persona, profesiones.
- Contiene sus propios filtros.
- Contiene modelos poseen dos modelos que serian el modelo de perfil de usuario perfil global y Tokens para realizar el restablecimiento de contraseñas y confirmar correo electrónico.
- Contiene tools algunas cosas son de forma global como los roles de usuario.

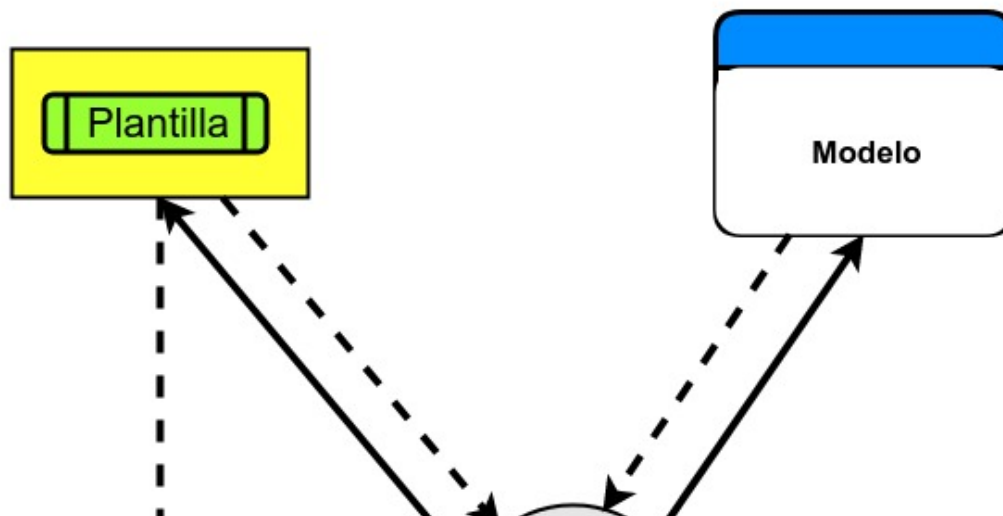
+ Projects: La aplicación principal para tipos de solicitud.

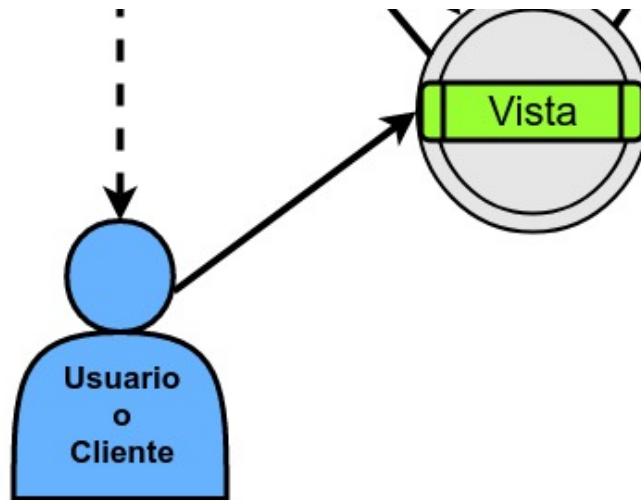
- Contiene la carpeta urls.
- Contiene la misma estructura que las demás aplicaciones.

Arquitectura MTV

Es el patrón de diseño empleado por Django donde:

- La M de “Model” o “Modelo”, es la capa de acceso a la base de datos. En esta capa se encuentra la información referentes a los datos y métodos asociados a ellos tales como: acceso, validación, comportamiento y relaciones.
- La T es por “Template” (en español plantilla), es la capa de presentación de la aplicación. Esta capa tiene las decisiones relacionadas a la presentación: tales como las que se mostrarán sobre una página web o cualquier otro tipo de documentos.
- La V es de “View” (Vista) , es la capa que alberga la lógica de negocios. Dentro de esta capa la lógica tiene acceso al modelo y delega a la plantilla: un ejemplo sería considerarlo como una especie puente entre el modelo y las plantillas.



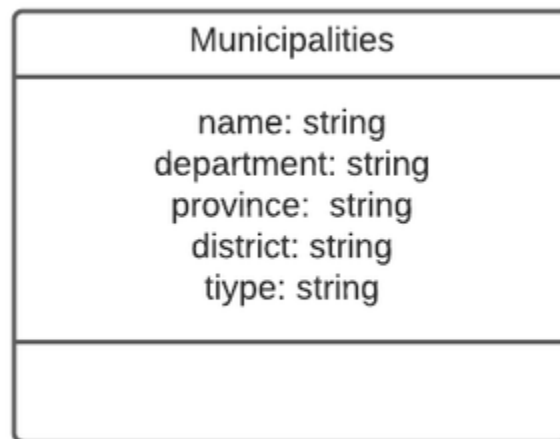


Si comparamos este patrón con el clásico MVC (Model-View-Controller), notamos que modelos es equivalente, lo que sería la “Vista” en el MVC pasa a ser el “Template” en el MVT y la capa del controlador, que es la capa que conecta al Modelo con la vista, en Django se puede considera que es el propio framework quién hace esta conexión, esto debido a que Django se encarga de manejar de manejo de las rutas y el enrutado.

Modelos

- **Dashboard:**

En toda la aplicación se hace uso del modelo de Municipalidades por eso se coloca en dashboard.



- **Evaluations:**

Se presentan los diagramas correspondientes a cada modelo:

- **Profiles:**

Se presentan los diagramas correspondientes a cada modelo:

- **Projects:**

Se presentan los diagramas correspondientes a cada modelo: