Prototipo de tablas de auditorías descentralizadas

Justificación

La librería Auditorías Turpial, cuenta con una única tabla para almacenar las trazas de auditoría de cada modelo considerado auditable. Dado que cada manejador de base de datos tiene un límite de cantidad de información que puede almacenar, esta aproximación evita que esta tabla crezca.

Es por esta razón que se decidió experimentar con la creación de un prototipo en el que no se posea una única tabla de auditoría, sino que cada modelo auditable tenga la suya. En este documento se presenta el diseño e implementación de la solución, así como las ventajas y desventajas que posee frente al modelo centralizado.

Diseño e implementación

La librería Auditorías Turpial utiliza para alguna de sus funcionalidades la *signals* provistas por Django. Al investigar respecto a estas, se descubrieron otras que permiten capturar otros eventos. En particular, esta solución se enfoca en el uso de la señal "class_prepared" qué es activada cuando un modelo está listo para ser utilizado.

Por otro lado, Python cuenta con la capacidad de crear clases de manera dinámica. Esto es beneficioso en caso de que se deseen usar técnicas como "metaprogramación". En particular, resulta útil para crear una clase cuya estructura interna, dependa de la ejecución del programa en cuestión. La función "type" puede recibir tres argumentos que permiten crear una clase:

- *classname*: es un *string* que define el nombre de la clase.
- *superclasses*: es una lista o tupla con las superclases de la que se está construyendo.
- *atributtes_dict*: es un diccionario que funciona como el *namespace* de la clase, agregando los atributos que indique con la especificación proporcionada.

Ambas ideas pueden combinarse para provocar el comportamiento que se deseaba lograr; obtener un modelo por cada tabla auditable. Con la *signal* "class_prepared" se puede detectar cuando un nuevo modelo auditable es agregado y con la función "type" se puede crear un la tabla de auditoría con los atributos necesarios. En la figura 1, se muestra un ejemplo del código en cuestión.

```python
@receiver(class_prepared)
def handler_class_prepared(sender, **kwargs):
    """Create an audit model for the specific class."""
    if issubclass(sender, AuditableMixin):
        name = str(sender.__name__ + 'Audit')

        class Meta:
            db_table = '%s_audit' % sender._meta.db_table
            app_label = 'auditor_core'

        # Set up a dictionary to simulate declarations within a class
        attrs = {
            '__module__': 'turpial_auditor.auditor_core.models',
            'Meta': Meta,
            'author': models.ForeignKey(settings.AUTH_USER_MODEL, null=True),
            'action': models.CharField(max_length=1, choices=ACTIONS_CHOICES),
            'datetime': models.DateTimeField(auto_now_add=True),
            'model_json_old': JSONField(),
            'model_json_new': JSONField(),
            '__str__': lambda self: auditToString(self)
        }

        model = type(name, (models.Model,), attrs)
```

Figura 1. Extracto del código para crear una tabla por modelo auditable.

Al nombre del modelo a auditar, se le agrega el sufijo “Audit” y al nombre de la base de datos se le añade “_audit” para diferenciarlo del que ya existe. Para este objeto, se agrega la superclase “models.Model” que es la que permite a Django considerarla un nuevo modelo y migrarlo. También, se agregan los mismos campos que poseen la tabla centralizada, exceptuando el “model_name” puesto que el nombre escogido para la clase ya indica a qué modelo pertenece.

El principal problema encontrado con esta solución, es una advertencia de que el modelo ya fue registrado.

Observaciones

- Si bien esta solución posee el comportamiento deseado, es necesario realizar una investigación más profunda para evitar la advertencia mencionada.
- A pesar de que descentralizar la información de las auditorías permite que la tabla no se sature tan rápidamente, generar varias tablas de auditoría puede tener una incidencia sobre el desempeño del sistema.