Repository Changes and Explanations

This document summarizes the code changes made to add Redis-backed caching and cache invalidation for property listings, and explains the purpose and behavior of each file.

properties/utils.py

Contains get_all_properties() which reads the cached "all_properties" key from Django cache, falls back to querying Property model, saves the evaluated list to cache for 3600 seconds, and returns it.

Also contains get_redis_cache_metrics() which connects to the redis instance used by django-redis, reads INFO, extracts keyspace_hits and keyspace_misses, computes a hit_rate (hits/(hits+misses)) and returns a dict. Any exceptions are logged via logger.error and the function returns None on error.

Source:

```
import logging
from django.core.cache import cache
from django_redis import get_redis_connection
from .models import Property

logger = logging.getLogger(__name__)


def get_all_properties():
"""Return all properties, using Redis cache key 'all_properties'.

The function first attempts to read the value from the cache. If the
value is missing, it fetches the queryset from the database, evaluates
it into a list to make it safe to cache, stores it in the cache for
3600 seconds (1 hour) and returns the list.
"""
cached = cache.get('all_properties')
if cached is not None:
return cached

qs = list(Property.objects.all().order_by('-created_at'))
cache.set('all_properties', qs, 3600)
return qs


def get_redis_cache_metrics():
"""Return Redis cache metrics (keyspace_hits, keyspace_misses, hit_rate).

Attempts to connect to the redis instance used by Django cache backend
(django-redis). On success returns a dict with integers for
'keyspace_hits' and 'keyspace_misses' and a float 'hit_rate' computed as
hits/(hits+misses) or 0 when there are no requests.

On error returns None and logs the exception via logger.error.
"""
try:
conn = get_redis_connection('default')
info = conn.info()
hits = int(info.get('keyspace_hits', 0))
misses = int(info.get('keyspace_misses', 0))
total_requests = hits + misses
hit_rate = (hits / total_requests) if total_requests > 0 else 0
return {
'keyspace_hits': hits,
'keyspace_misses': misses,
'hit_rate': hit_rate,
}
except Exception as e:
```

```
logger.error('Failed to get redis metrics: %s', e)
return None
```

properties/views.py

Defines PropertyListView (class-based) used for HTML listing and property_list (function-based) which returns JSON and is decorated with @cache_page(60*15) so the HTTP response is cached for 15 minutes. The property_list view uses get_all_properties() from utils to pull cached DB results.

Source:

```
from django.shortcuts import render
from django.views.generic import ListView
from django.views.decorators.cache import cache_page
from django.utils.decorators import method_decorator
from django.http import JsonResponse
from .models import Property
from .utils import get_all_properties


@method_decorator(cache_page(60 * 15), name='dispatch') # Cache for 15 minutes
class PropertyListView(ListView):
model = Property
template_name = 'properties/property_list.html'
context_object_name = 'properties'

def get_queryset(self):
"""Return all properties ordered by creation date."""
return Property.objects.all().order_by('-created_at')


@cache_page(60 * 15)
def property_list(request):
"""Function-based view that returns all properties as JSON.

Response cached in Redis for 15 minutes by the @cache_page decorator.
Returns JSON in the shape: {"data": [ {property fields...}, ... ] }
"""
qs = get_all_properties()
data = []
for p in qs:
data.append({
'id': p.id,
'title': p.title,
'description': p.description,
'price': str(p.price),
'location': p.location,
'bedrooms': p.bedrooms,
'bathrooms': p.bathrooms,
'square_feet': p.square_feet,
'property_type': p.property_type,
'status': p.status,
'created_at': p.created_at.isoformat() if p.created_at else None,
'updated_at': p.updated_at.isoformat() if p.updated_at else None,
})

return JsonResponse({'data': data})
```

properties/signals.py

Provides post_save and post_delete signal handlers that delete the cache key "all_properties" when Property instances change. This prevents stale cached DB results.

Source:

```
from django.core.cache import cache
from django.db.models.signals import post_save, post_delete
from django.dispatch import receiver
from .models import Property


@receiver(post_save, sender=Property)
def clear_all_properties_cache_on_save(sender, instance, **kwargs):
"""Clear the 'all_properties' cache key when a Property is saved."""
cache.delete('all_properties')


@receiver(post_delete, sender=Property)
def clear_all_properties_cache_on_delete(sender, instance, **kwargs):
"""Clear the 'all_properties' cache key when a Property is deleted."""
cache.delete('all_properties')
```

properties/apps.py

Defines PropertiesConfig and includes an import of properties.signals to ensure signal handlers are registered. An explicit import at module level was also added to guarantee the presence of the import.

Source:

```
from django.apps import AppConfig
import properties.signals # ensure signals module is importable at module import
time


class PropertiesConfig(AppConfig):
default_auto_field = 'django.db.models.BigAutoField'
name = 'properties'

def ready(self):
# Import signal handlers to connect them. Placing import here
# ensures the app registry is ready and avoids import-time side effects.
from . import signals # noqa: F401
```

properties/__init__.py

Sets default_app_config to properties.apps.PropertiesConfig for compatibility with certain Django versions.

Source:

```
default_app_config = 'properties.apps.PropertiesConfig'
```