data-logging

Выпуск

BARS Group

07 May 2015

Оглавление

Version 1.2.2

Data Logging - логгер действий пользователя в системе. Отслеживаются такие события, как вход, выход пользователя, открытие окон, удаление, редактирование, создание записей в БД.

Примечание: Не реализована поддержка NoSQL решений.

Внимание: Массовые операции (Model.objects.delete() и т.д.) над моделями не отслеживаются.

Глава 1

Зависимости

M3 1.5

Django 1.3

Django JSON Field

South

Architect

Глава 2

Настройки

В `settings.DATABASE_ROUTERS` необходимо добавить `datalogging.dbrouters.DataLoggerRouter`. Если используется отдельная БД так же необходимо добавить `datalogging.dbrouters.NotUseDataLoggerDBRouter`.

В `settings.MIDDLEWARE_CLASSES` необходимо добавить `datalogging.middleware.CaptureRequestMiddleware` и `datalogging.middleware.RequestTokenMiddleware`.

`DATALOGGER_EVENT_TYPE` Разделение событий по типу (системные, юридически важные и т.д.). В дальнейшем, к перечислению можно обратиться через `EventType`.

Пример:

```
DATALOGGER_EVENT_TYPE = {
    'SYSTEM': ('se', u'Системное событие'),
}
```

`DATALOGGER_EVENT_CODE` Коды событий, изначально имеются 6 событий: insert, update, delete, login, logout, win_open. В дальнейшем, к перечислению можно обратиться через `EventCode`.

Пример:

```
DATALOGGER_EVENT_CODE = {
    'CUSTOM_EVENT_CODE': ('cec', 'Собственный код события'),
}
```

`DATALOGGER_SYSTEMS_ENUM` Перечисление подсистем, в которых возникает событие: В дальнейшем, к перечислению можно обратиться через `SystemEnum`.

Пример:

```
DATALOGGER_SYSTEMS_ENUM = {
    'APPICATION': ('app', u'Основное приложение'),
}
```

`DATALOGGER_SUSPECTS_MODEL` Список моделей, состояние которых необходимо отслеживать. Причем, в качестве элемента списка можно использовать как полный путь до модели, так и сокращенный. Как альтернатива, можно указывать DataLoggingManager прямо в классе модели.

Пример:

```
DATALOGGER_SUSPECTS_MODEL = [
    'module_name.ModelName',
    'project_name.core.module_name.models.ModelName',
]
```

`DATALOGGER_EXCLUDE_ACTIONS` Перечисление имен классов паков или экшенов, которые не нужно логировать.

DATALOGGER_EXCLUDE_FIELDS Перечень полей, которые не надо использовать при сравнении состояний модели.

Пример:

```
DATALOGGER_EXCLUDE_FIELDS = [
    'version', 'begin', 'end', 'modified',
]
```

DATALOGGER_DATABASE Наименование БД из settings.DATABASES, которая будет использована в качестве бэкенда для логгера.

В случае, если БД отлична от основной БД приложения, то необходимо включить в settings.DATABASE_ROUTERS классы NotUseDataLoggerDBRouter и DataLoggerRouter.

Для корректной работы с миграциями требуется подключить модуль datalogging.south_migration который перекрывает команду South - migrate.

Примечание: Так же требуется предварительный запуск команды syncdb.

Пример:

```
INSTALLED_APPS += ('datalogging.south_migration',)

DATALOGGER_DATABASE = 'log_db'

DATABASE_ROUTERS = (
    'datalogging.dbrouters.DataLoggerRouter',
    'datalogging.dbrouters.NotUseDataLoggerDBRouter',
)
```

DATALOGGER_SHUTUP Если необходимо отключить логгер, то значение должно быть True в ином случае False.

DATALOGGER_FORGET_SYS_EVENTS Если необходимо отключить логирование системных событий, то значение должно быть True в ином случае False.

DATALOGGER_HOOKED_ACTIONS Для возможности кастомного логирования вызова определенных экшенов, требуется указать их в словаре вида:

```
DATALOGGER_EVENT_CODE = {
    'CUSTOM_EVENT_CODE': ('cec', 'description event')
}

DATALOGGER_HOOKED_ACTIONS = {
    'SomeActionClassName': 'CUSTOM_EVENT_CODE'
}
```

Позже, событие с экшеном можно перехватить по коду в обработчике сигнала post_system_event_signal, в kwargs будут присутствовать action и request.

DATALOGGER_COMPRESS_FILENAME_TEMPLATE Определяет формат именования файлов при архивировании логов. По умолчанию datalogging-dump-%d-%m-%Y-%H-%M.

DATALOGGER_COMPRESS_DESTINATION Определяет место назначения для архивов лога. По умолчанию: текущая папка.

Глава 3

Использование

Добавить в `INSTALLED_APPS`.

Для использования логгера строго необходимо определить обработчики сигналов.

3.1 Сигналы

`msg_for_log_signal` Сигнал возникает при формировании логгером человекпонятного описания события. В случае, если событие системное(открытие пользователем окошка), то `model_instance` будет иметь `None` в качестве значения. В качестве возвращаемого значения должна быть представлена строка.

Передаваемые аргументы:

- `log_record` - экземпляр записи лога,
- `model_instance` - экземпляр записи модели,
- `fields_dict` - словарь полей экземпляра модели, где ключ - имя поля, а значение - значение поля в модели.

`log_context_signal` Если приложение запущено в режиме фоновой задачи (Celery и т.д.) или в режиме шела, то `request` будет `None`.

Контекст события должен являться словарем и содержать значения:

- `suid` - ID пользователя в среде, в которой произошло событие,
- `system_type` - значение из `SystemEnum` описывающее текущую среду,
- `event_type` - значение из `EventType` описывающее текущий тип события.

В качестве возвращаемого значения должен быть представлен словарь.

Передаваемые аргументы:

- `request` - текущий запрос.

`post_snapshot_signal` Вызывается в момент формирования записи об измененном состоянии отслеживаемой модели. Позволяет изменить запись лога перед сохранением.

Передаваемые аргументы:

- `log_record` - не сохраненный в БД экземпляр записи лога

`post_system_event_signal` Вызывается в момент формирования записи о событии происшедшем в системе. Позволяет изменить запись лога перед сохранением.

Передаваемые аргументы:

- `log_record` - не сохраненный в БД экземпляр записи лога

3.2 Выборка записей

`filter_events` Позволяет отфильтровать записи лога. По поведению функция схожа с методом `filter` в Django ORM, с той лишь разницей, что есть возможность осуществлять поиск по сериализованным в JSON данным.

Пример (поиск по загловку окна):

```
filter_events(
    event_code=EventCode.WIN_OPEN,
    _context__title=u'Заголовок или его часть')
```

`get_events_by_token` Позволяет получить все записи с одинаковым токеном запроса. Т.е. все события возникшие в рамках одного запроса.

Пример:

```
get_events_by_token(some_log_record.request_token)
```

3.3 Партицирование таблицы хранения логов по месяцам

Для включения партицирования используйте manage команду $python manage.py datalogging_partition_prepare

3.4 Пример

settings.py

```
...

DATALOGGER_DATABASE = 'default'
DATALOGGER_EXCLUDE_FIELDS = ('version', 'modified')

DATALOGGER_EVENT_CODE = {
    'CRITICAL_CHANGE': ('cc', u'Критичное изменение'),
}

DATALOGGER_SYSTEMS_ENUM = {
    'APPLICATION': ('app', u'Основное приложение'),
    'SCHEDULER': ('sch', u'Задачи вызыванные планировщиком'),
}

DATALOGGER_EVENT_TYPE = {
    'SYSTEM_EVENT': ('se', u'Системное событие'),
    'LEGALLY_EVENT': ('le', u'Юридически важное событие'),
}
```

signals.py

```
from datalogging.signals import custom_verbose, custom_log_context, post_snapshot


def verbose_handler(sender, log_record, model_instance, fields_dict):
    model_mapping = {
        'module.Declaration': u'заявку (ID=%(id)s)',
        'module.DeclarationUnit': u'привязку заявки (ID=%(declaration_id)s) к учреждению (ID=%(unit_id)s)',
        'module.Children': u'ребенка (ID=%(id)s)',
        'module.Pupil': u'запись о зачислении ребенка (ID=%(children_id)s в группу (ID=%(grup_id)s)',
        'module.Deduct': u'запись об отчислении ребенка (ID=%(children_id)s) из группы (ID=%(group_id)s)',
        'module.Group': u'группу (ID=%(id)s) учреждения (ID=%(unit_id)s)',
        'module.Direct': u'направление заявки %(declaration_id)s в группу %(group_id)s'
    }

    operation_mapping = {
        EventCode.UPDATE: u'изменил(а)',
        EventCode.INSERT: u'создал(а)',
        EventCode.DELETE: u'удалил(а)'
    }

    if log_record.event_code == EventCode.WIN_OPEN:
        return u'Открыто окно: %s' % log_record.context_data['title']

    what = u'запись в "%s"' % model_instance._meta.verbose_name
    if log_record.object_type in model_mapping:
        what = model_mapping[log_rec.object_type] % fields_dict

    verbose = u'Пользователь (ID=%s) %s %s.' % (
        log_record.suid,
        operation_mapping.get(log_rec.event_code),
        what
    )

    return verbose


def context_handler(sender, request):
    if request is None:
        user_id = None
        system_type = SystemsEnum.SHELL
        event_type = EventType.UNDEFINED
    else:
        user_id = getattr(request.user, 'id', None)
        url = request.get_full_path()
        if '/some_pattern' in url:
            event_type = EventType.SOME_TYPE
            system_type = SystemsEnum.APPLICATION
        elif '/some_diffierent_pattern' in url:
            event_type = EventType.SOME_DIFFERENT_TYPE
            system_type = SystemType.SCHEDULER

    return {'suid': user_id,
            'event_type': event_type,
            'system_type': system_type}


def post_snapshot_handler(sender, log_record):
```

```
    if log_record.object_type == 'module.SomeModelName' and log_record.event_code = EventCode.UPDATE:
        log_record.event_code = EventCode.CRITICAL_CHANGE


msg_for_log_signal.connect(verbose_handler, weak=False)
log_context_signal.connect(context_handler, weak=False)
post_snapshot_signal.connect(post_snapshot_handler, weak=False)
```

Contents:

3.4.1 Описание

3.4.2 Установка

3.4.3 Демо проект

3.4.4 Простое использование

3.4.5 Состав модуля

Логгер действий пользователя в системе.

datalogging	Логгер действий пользователя в системе.
datalogging.api	
datalogging.app_meta	
datalogging.dbrouters	
datalogging.enums	
datalogging.helpers	
datalogging.management.commands.datalogging_compress	
datalogging.middleware	
datalogging.models	
datalogging.signals	

api

datalogging.api.filter_events(***kw*)

Фильтрация по записям лолга.

Примечание: Не предусмотрена работа с NoSQL.

Если необходимо произвести фильтрацию по данным находящимся в в полях `context_data` или `object_snapshot`. То необходимо использовать префиксы `_context` и `_snapshot` соответственно.

```
    filter_events(
        object_type='enterprise.Enterprise',
        system_type=SystemsEnum.MAIN_APPLICATION,
        _context__diff__name='Учреждение №666',
        _context__ent='455',
    )
```

datalogging.api.get_events_by_token(*request_ token*)

Получение записей лога по их общему токену

datalogging.api.get_request()
Получение текущего запроса.

datalogging.api.get_request_token()
Получение токена текущего запроса.

datalogging.api.get_user_ip()
Получение IP текущего пользователя.

Return str or None Если не удалось получить IP вернет None

app_meta

class datalogging.app_meta.InvaderLogPack
Инжектируемый пак

post_run(*request*, *context*, *response*)
Некоторые косвенные методы определения поведения пользователя. Если в ответе на запрос упоминается заголовок, то считается, что вернули окно. Если запомненный ранее пользователь был аутентифицирован, а после формирования ответа - нет, то считается, что произошел выход из системы. И наоборот.

pre_run(*request*, *context*)
Перед выполнением экшена у инжектируемого пака необходимо запоминить пользователя

datalogging.app_meta.invader_invoke(*self*, *request*, *action*, *stack*, **args*, ***kwargs*)

Параметры

- request (*request*) – Экземпляр Action'a.
- action (*Action*) – Экземпляр Action'a.
- stack (*list*) – Экземпляр Action'a.

datalogging.app_meta.is_loggable(*obj*)
Данная функция использоуется для опредения нуждается ли данный объект в логировании. Проводится проверка объекта на вхождение имени его класса, либо имени класса его предка в список-исключение *EXCLUDED_ACTIONS*. Если переданный экземпляр найден в списке, то объект считается подходящим для создания записи.

Параметры obj (*Action*) – Экземпляр Action'a.

Результат bool – результат проверки.

dbrouters

class datalogging.dbrouters.DataLoggerRouter
Роутер не позволяет создавать связи с моделью лога.

allow_relation(*obj1*, *obj2*, ***hints*)
Метод контролируюий факт создания связи между обектами. Отсекаются связи создаваемые с моделяи данного модуля.

allow_syncdb(*db*, *model*)
Метод регулирующий возможность инициализации моделей.

db_for_read(*model*, ***hints*)
Выбор БД для чтения. В данном методе происходит проверка на заданное имя модели и в случаем если модель принадлежит данному модулю, для нее используются отдельные настройки

db_for_write(*model*, ***hints*)
Выбор БД для записи. Применяются те же правила, что и при чтения из базы.

class datalogging.dbrouters.NotUseDataLoggerDBRouter
Роутер запрещает создание таблиц в базе для логгера.

allow_syncdb(*db*, *model*)
Запрет создания таблиц в базе логгера

enums

class datalogging.enums.ConfigurableEnum
Метакласс автозагрузки типов перечислений.

Автозагрузка типов осуществляется из settings.py проекта, причем данные должны являться словарем с кортежами в качестве значений. Наименование параметра должно начинаться с приставки "DATALOGGER". Пример:

```
DATALOGGER_EVENT_TYPE = {
    'SYSTEM': ('se', u'Системное событие'),

}
```

После инициализации класса, можно вызывать как обычный аттрибут. И в целом класс используется как обычное перечисление. Пример:

```
print EventType.SYSTEM // 'se'
```

class datalogging.enums.EventCode
Перечисление кодов событий.

Пример возможных кодов: - (insert, Добавление) - (delete, Удаление)

class datalogging.enums.EventType
Перечисление типов событий.

Пример возможных типов: - (se, Системное событие) - (lse, Юридически важное событие)

class datalogging.enums.SystemsEnum
Перечисление сред логирования.

Пример возможных значений: - ('scheduled_task', u'Задача по расписанию') - ('ws', u'Web-сервис')

helpers

datalogging.helpers.get_snapshot(*instance*)
Получение снимка состояния экземпляра модели

datalogging.helpers.is_server_mode()
Определение режима работы сервера.

datalogging.helpers.memorize_user(*user*)
Запоминание пользователя в текущем треде.

datalogging.helpers.only_server_mode(*func*)
Предотвращение запуска логгера в шеле

datalogging.helpers.remember_user()
Вспоминаем пользователя

managers

middleware

class `datalogging.middleware.CaptureRequestMiddleware`
Middleware сохраняющее текущий request.

В последующем request используется для получения ряда параметров необходимых для логирования действий (например ip пользователя).

class `datalogging.middleware.RequestTokenMiddleware`
Middleware создающее токен для текущего запроса.

Токен в последующем устаналвивается во все возникающие события в логгере, таким образом можно просмотреть/отследить цепочку событий возникших во время запроса.

model

class `datalogging.models.DataLog(`**args*, ***kwargs*`)`
Лог активности пользователей.

Лог предусматривает отслеживание активности пользователей в системе, такой, как удаление/добавление/изменение записей документов, настро ек, прав доступа и т.д.

***guid*:** Уникальный ИД записи не является primary_key в случае с реляционными БД. Его необходимость возникает в момент выгрузки, когда несколько логов выгружаются в одну систему и применение primary_key из реляционной БД вызвало бы ряд неудобств связанных с коллизиями. Во избежание коллизий используется uuid с солью.

***system_type*:** В качестве подсистемы выступают сервисы(rest, soap), фоновые задачи, основное приложение и т.д. Т.е. это по существу среда возникновения события, которое следует перехватить.

***request_token*:** В рамках одного запроса сущетствует токен по которому можно отследить всю активность пользователя в течении запроса.

classmethod `get_verbose_msg(`*log_record*, *model_instance*, ***kwargs*`)`
Получение человекопонятного описания события.

Т.к. для каждого проекта формирование описания будет разным метод работает через сигнал *msg_for_log_signal*.

classmethod `make_snapshot(`*instance*, *event_code*`)`
Формирование записи лога о состоянии модели.

Перед сохранением записи лога вызывается сигнал *post_snapshot*, где в качестве аргумента передается не сохраненная запись лога, что в свою очередь позволяет воздействовать на запись.

Параметры

- `instance` – Экземпляр модели
- `event_code` – Событие происшедшее с моделью

classmethod `make_system_event(`*event_code*, *context_data=None*, ***kwargs*`)`
Формирование записи о событии происшедшем в системе.

Все спецефичные данные о событии обрабатываются через сигнал *post_system_event* ровно так же, как в методе *make_snapshot*.

Параметры

- `event_code` – Событие в системе.
- `context_data` (*dict*) – Доп. данные описывающие событие.

`save`(*instance*, **args*, ***kwargs*)
Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

class `datalogging.models.LogJSONEncoder`(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)
Наследник JSONEncoder, который умеет работать с ugettext.

signals

`datalogging.signals.add_datalogger_manager`(*sender*, ***kw*)
Обработчик сигнала class_prepared.

Если модель указана в списке `DATALOGGER_SUSPECTS_MODEL`, то обработчик добавляет в класс модели `DataLoggingManager`.

datalogging_compress

class `datalogging.management.commands.datalogging_compress.Command`
Класс менежд-команды.

`handle`(**args*, ***options*)
Метод обрабатыющий команду. На входе ожидаются только именованые параметры переданные из командной строки, в частности *options['reset']* и *options['destination']*.

Indices and tables

- genindex
- modindex
- search

d

A

C

D

E

F

G

H

I

L

M