xlsx$_{streaming}$

Release 1.2.0.dev0

Jun 23, 2021

Contents

`xlsx_streaming` is a library to stream an xlsx_document from a collection of rows of data (typically a queryset containing database data).

Contents:

CHAPTER 1

First steps

The `xlsx_streaming` library is primarily meant to be used with streaming HTTP response objects. It provides an API to stream data extracted from a database directly into an xlsx document.

This library does not depend on any web framework or any python excel library. However it was primarily designed for django and has not been tested with any other web framework yet.

1.1 Django example

```
from django.http import StreamingHttpResponse

import xlsx_streaming

def my_view(request):
    qs = MyModel.objects.all().values_list('field1', 'field2', 'field3')
    with open('template.xlsx', 'rb') as template:
        stream = xlsx_streaming.stream_queryset_as_xlsx(
            qs,
            template,
            batch_size=50
        )

    response = StreamingHttpResponse(
        stream,
        content_type='application/vnd.xlsxformats-officedocument.spreadsheetml.sheet',
    )
    response['Content-Disposition'] = 'attachment; filename=test.xlsx'
    return response
```

With this kind of code, your database will not be hit too hard by the file export, even when the export's size is several dozens of megabytes.

To include type information (for dates, numbers, ...) in the generated Excel file, `xlsx_streaming` uses a user-provided template file containing example data whose column types match those of the exported data. Like in the

example above, you can use a static template, but you can also generate the template dynamically using the python excel library of your choice (openpyxl, ...). The template must be a valid Excel file with at least one row. If the template contains only one row, this row cell's datatypes are used as datatypes for all generated rows. If the template contains more than one row, the first row is kept as is (header row) and the second row cell's datatypes are used as datatypes for all generated rows. The template must be open as binary.

CHAPTER 2

Built-in serialization

The `xlsx_streaming` library provides builtin support for numerical, date/datetime/timedelta, and boolean conversion to excel equivalent. All other data types are converted to text by default.

CHAPTER 3

Using custom serializers

If you want to preprocess the data from the database before inserting it in the xlsx document, you can pass a serializer argument to `stream_queryset_as_xlsx`. This function is applied to each queryset slice before writing the data to excel. For example, with a Django Rest Framework serializer:

```
class MySerializer(serializers.Serializer):
    ...

serializer = lambda x: [d.values() for d in MySerializer(x, many=True).data]
xlsx_streaming.stream_queryset_as_xlsx(qs, template, serializer=serializer)
```

CHAPTER 4

Specifying the timezone of the export

The datetime objects might be stored as 'UTC' inside your database, but you may want to see datetime objects exported in a specific timezone in the excel document. Excel does not offer support for timezone. However, the `xlsx_streaming` library lets you specify the timezone to be used for exports (by default the timezone is kept unchanged). Before writing a datetime, it will first be localized in the specified timezone before being converted to a naive datetime. This functionality requires the pytz library.

For example, if you set:

```
from pytz import timezone

xlsx_streaming.set_export_timezone(timezone('US/Eastern'))
```

all the datetimes written with the `xlsx_streaming` library will first be localized in the 'US/Eastern' timezone.

CHAPTER 5

Reference

xlsx_streaming.**stream_queryset_as_xlsx**(*qs*, *xlsx_template=None*, *serializer=None*, *batch_size=1000*, *encoding='utf-8'*)

Iterate over qs by batch (typically a Django queryset) and stream the bytes of the xlsx document generated from the data.

This function can typically be used to stream data extracted from a database by batch, making many small database requests instead of one big request which could timeout.

Parameters

- **qs** (*Iterable*) – an iterable containing the rows (typically a Django queryset)
- **xlsx_template** (*Optional[BytesIO]*) – an in memory xlsx file template containing the header (optional) and the first row used to infer data types for each column. If not provided, all cells will be formatted as text.
- **serializer** (*Optional[Callable]*) – a function applied to each batch of rows to transform them before saving them to the xlsx document (defaults to identity).
- **batch_size** (*Optional[int]*) – the size of each batch of rows
- **encoding** (*Optional[str]*) – the file encoding

Returns A streamable xlsx file

Return type Iterable

Note: If the xlsx template contains more than one row, the first row is kept as is in the final xlsx file (header row), and the second one is used as a template for all the generated rows.

CHAPTER 6

Changelog

6.1 1.2.0 (unreleased)

- Nothing changed yet.

6.2 1.1.0 (2021-06-23)

- `render_worksheet()` now allows rendering worksheet with iterators.

6.3 1.0.0 (2021-03-01)

- `stream_queryset_as_xlsx()` now accepts an iterator.
- Drop support of Python 2, 3.4 and 3.5. xlxs_streaming now supports only Python 3.6 and onward.

6.4 0.4.1 (2019-11-07)

- Fix bug in `stream_queryset_as_xlsx()` that was introduced in version 0.4.0. When called with a Django QuerySet object (or something similar), the function made a single SQL query, instead of multiple SQL queries depending on the `batch_size` argument. This bug may cause performance issues when fetching many rows from the database.

6.5 0.4.0 (2019-11-06)

- `stream_queryset_as_xlsx()` now accepts a generator. [Hugo Lecuyer].

6.6 0.3.1 (2017-02-22)

- First sheet of a workbook is now more reliably detected

6.7 0.3.0 (2017-02-22)

- It is now possible to stream data to an Excel document without providing a template. In this case all cells are formatted as text (no date or number formatting). If there is an error with the template provided by the user, fall back to this behaviour.
- Improve error handling (try to recover and log instead of raise)
- Change license

6.8 0.2.0 (2015-12-16)

- Add python 2.6 compatibility

6.9 0.1.0 (2015-12-16)

- First public version.

CHAPTER 7

Indices and tables

- genindex
- search

Index

S