django-excel Documentation

Release 0.0.10

Onni Software Ltd.

Nov 10, 2020

Contents

Author C.W.

Source code http://github.com/pyexcel-webwares/django-excel.git

Issues http://github.com/pyexcel-webwares/django-excel/issues

License New BSD License

Released 0.0.10

Generated Nov 10, 2020

Here is a typical conversation between the developer and the user:

```
User: "I have uploaded an excel file"
      "but your application says un-supported file format"
Developer: "Did you upload an xlsx file or a csv file?"
User: "Well, I am not sure. I saved the data using "
      "Microsoft Excel. Surely, it must be in an excel format."
Developer: "OK. Here is the thing. I were not told to support"
           "all available excel formats in day 1. Live with it"
           "or delay the project x number of days."
```

django-excel is based on pyexcel and makes it easy to consume/produce information stored in excel files over HTTP protocol as well as on file system. This library can turn the excel data into a list of lists, a list of records(dictionaries), dictionaries of lists. And vice versa. Hence it lets you focus on data in Django based web development, instead of file formats.

The idea originated from the common usability problem: when an excel file driven web application is delivered for non-developer users (ie: team assistant, human resource administrator etc). The fact is that not everyone knows (or cares) about the differences between various excel formats: csv, xls, xlsx are all the same to them. Instead of training those users about file formats, this library helps web developers to handle most of the excel file formats by providing a common programming interface. To add a specific excel file format type to you application, all you need is to install an extra pyexcel plugin. Hence no code changes to your application and no issues with excel file formats any more. Looking at the community, this library and its associated ones try to become a small and easy to install alternative to Pandas.

The highlighted features are:

1. excel data import into and export from databases
2. turn uploaded excel file directly into Python data structure
3. pass Python data structures as an excel file download
4. provide data persistence as an excel file in server side
5. supports csv, tsv, csvz, tsvz by default and other formats are supported via the following plugins:

Table 1: A list of file formats supported by external plugins

Package name	Supported file formats	Dependencies
pyexcel-io	csv, csvz[1], tsv, tsvz[2]	
pyexcel-xls	xls, xlsx(read only), xlsm(read only)	xlrd, xlwt
pyexcel-xlsx	xlsx	openpyxl
pyexcel-ods3	ods	pyexcel-ezodf, lxml
pyexcel-ods	ods	odfpy

[1] zipped csv file
[2] zipped tsv file

Table 2: Dedicated file reader and writers

Package name	Supported file formats	Dependencies
pyexcel-xlsxw	xlsx(write only)	XlsxWriter
pyexcel-libxlsxw	xlsx(write only)	libxlsxwriter
pyexcel-xlsxr	xlsx(read only)	lxml
pyexcel-xlsbr	xlsb(read only)	pyxlsb
pyexcel-odsr	read only for ods, fods	lxml
pyexcel-odsw	write only for ods	loxun
pyexcel-htmlr	html(read only)	lxml,html5lib
pyexcel-pdfr	pdf(read only)	camelot

CHAPTER 1

Plugin shopping guide

Since 2020, all pyexcel-io plugins have dropped the support for python version lower than 3.6. If you want to use any python verions, please use pyexcel-io and its plugins version lower than 0.6.0.

Except csv files, xls, xlsx and ods files are a zip of a folder containing a lot of xml files

The dedicated readers for excel files can stream read

In order to manage the list of plugins installed, you need to use pip to add or remove a plugin. When you use virtualenv, you can have different plugins per virtual environment. In the situation where you have multiple plugins that does the same thing in your environment, you need to tell pyexcel which plugin to use per function call. For example, pyexcel-ods and pyexcel-odsr, and you want to get_array to use pyexcel-odsr. You need to append get_array(..., library='pyexcel-odsr').

Table 1: Other data renderers

Package name	Supported file formats	Dependencies	Python versions
pyexcel-text	write only:rst, mediawiki, html, latex, grid, pipe, orgtbl, plain simple read only: ndjson r/w: json	tabulate	2.6, 2.7, 3.3, 3.4 3.5, 3.6, pypy
pyexcel-handsontable	handsontable in html	handsontable	same as above
pyexcel-pygal	svg chart	pygal	2.7, 3.3, 3.4, 3.5 3.6, pypy
pyexcel-sortable	sortable table in html	csvtotable	same as above
pyexcel-gantt	gantt chart in html	frappe-gantt	except pypy, same as above

This library makes information processing involving various excel files as easy as processing array, dictionary when processing file upload/download, data import into and export from SQL databases, information analysis and persistence. It uses **pyexcel** and its plugins:

1. to provide one uniform programming interface to handle csv, tsv, xls, xlsx, xlsm and ods formats.

2. to provide one-stop utility to import the data in uploaded file into a database and to export tables in a database as excel files for file download.
3. to provide the same interface for information persistence at server side: saving a uploaded excel file to and loading a saved excel file from file system.

Given the existence of pyexcel, what is the reason for django-excel? 1. **Speedy file uploads**. **django-excel** help you access the uploaded excel file directly using ExcelMemoryFileUploadHandler and TemporaryExcelFileUploadHandler. MemoryFileUploadHandler holds the uploaded file in memory and django-excel reads the excel data from this memory buffer without caching it onto file system. Meanwhile, TemporaryExcelFileUploadHandler holds the uploaded file in file system and django-excel reads directly from this stream-to-file without extra function calls. 2. **Import excel data into database**. **django-excel** uses bulk_insert to import your excel data into your django Model, which is very efficient.

CHAPTER 2

Installation

You can install django-excel via pip:

```
$ pip install django-excel
```

or clone it and install it:

```
$ git clone https://github.com/pyexcel-webwares/django-excel.git
$ cd django-excel
$ python setup.py install
```

Installation of individual plugins , please refer to individual plugin page. For example, if you need xlsx file support, please install pyexcel-xlsx:

```
$ pip install pyexcel-xlsx
```

Contrary to Django's philosophy of 'battery included', django-excel does not come with all batteries due to the size of the dependency(xlwt, openpyxl, odfpy). Hence, Django developer is left with the choice to install and load the excel file formats.

CHAPTER 3

Setup

You will need to update your *settings.py*:

```
FILE_UPLOAD_HANDLERS = ("django_excel.ExcelMemoryFileUploadHandler",
                        "django_excel.TemporaryExcelFileUploadHandler")
```

CHAPTER 4

Tested Django Versions

2.1, 2.08, 1.11.15, 1.10.8, 1.9.13, 1.8.18, 1.7.11, 1.6.11

Since 15 March 2015, python 2.6 are no longer tested via travis-ci.

CHAPTER 5

Support the project

If your company has embedded pyexcel and its components into a revenue generating product, please support me on github, patreon or bounty source to maintain the project and develop it further.

If you are an individual, you are welcome to support me too and for however long you feel like. As my backer, you will receive early access to pyexcel related contents.

And your issues will get prioritized if you would like to become my patreon as *pyexcel pro user*.

With your financial support, I will be able to invest a little bit more time in coding, documentation and writing interesting posts.

CHAPTER 6

More excel file formats

The example application understands csv, tsv and its zipped variants: csvz and tsvz. If you would like to expand the list of supported excel file formats (see *A list of file formats supported by external plugins*) for your own application, you could install one or all of the following:

```
pip install pyexcel-xls
pip install pyexcel-xlsx
pip install pyexcel-ods
```

Warning: If you are using pyexcel <=0.2.1, you still need to import each plugin manually, e.g. *import pyexcel.ext.xls* and Your IDE or pyflakes may highlight it as un-used but it is used. The registration of the extra file format support happens when the import action is performed

CHAPTER 7

Tutorial

In order to dive in django-excel and get hands-on experience quickly, the test application for django-excel will be introduced here. So, it is advisable that you should check out the code from github

```
git clone https://github.com/pyexcel/django-excel.git
```

The test application is written according to Part 1, Part 2 and Part 3 of django tutorial. If you should wonder how the test application was written, please visit django documentation and come back.

Once you have the code, please change to django-excel directory and then install all dependencies:

```
$ cd django-excel
$ pip install -r requirements.txt
$ pip install -r tests/requirements.txt
```

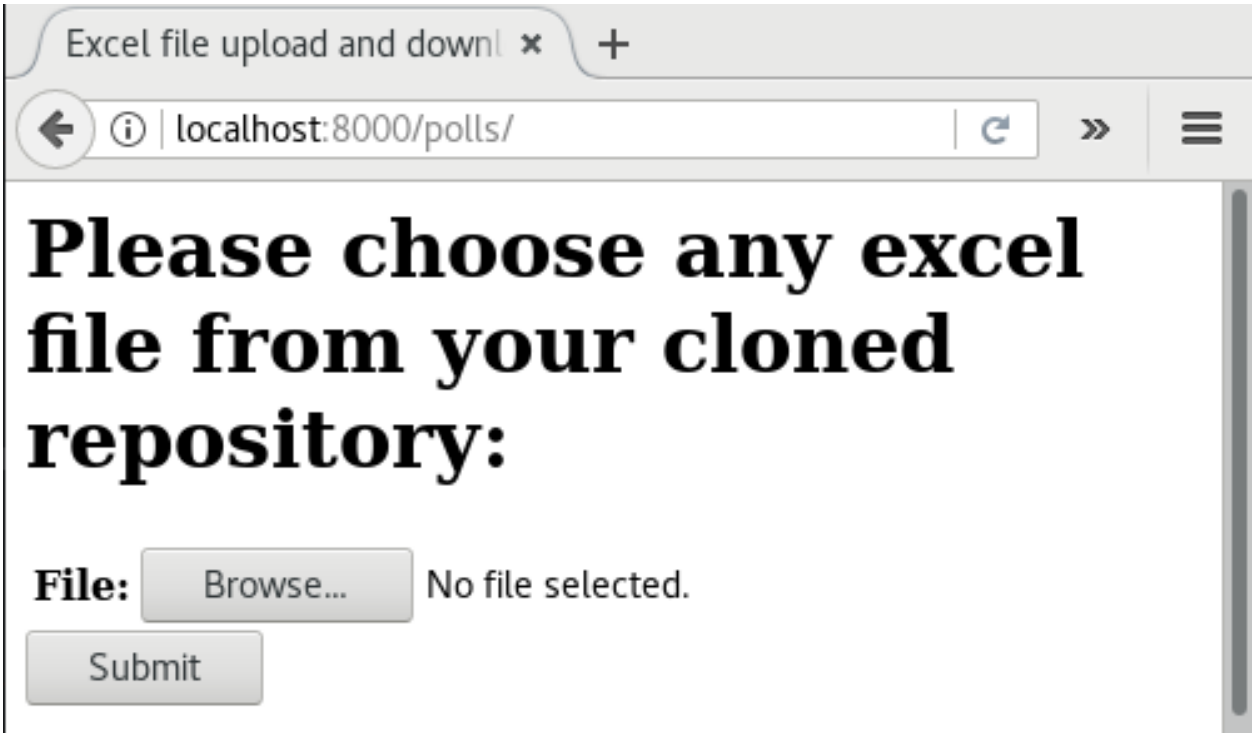
Then run the test application:

```
$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).

You have 9 unapplied migration(s). Your project may not work properly until you apply␣
↪the migrations for app(s): admin, auth, contenttypes.
Run 'python manage.py migrate' to apply them.

July 06, 2017 - 08:29:10
Django version 1.11.3, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

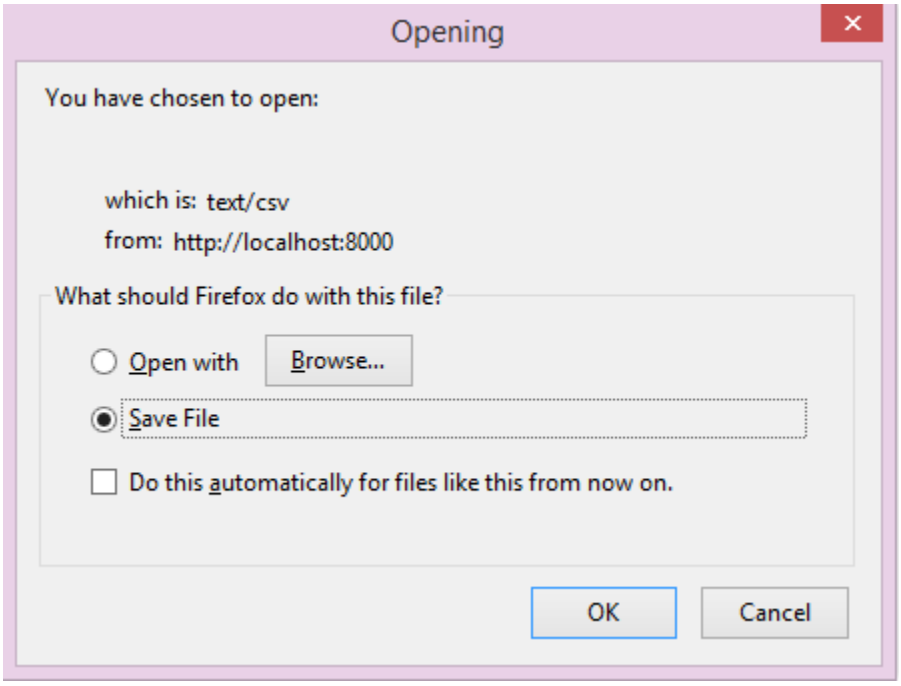
Note: The 9 unapplied migration(s) were ignored because migrations are out of scope in this tutorial.

7.1 Handle excel file upload and download

This example shows how to process uploaded excel file and how to make data download as an excel file. Open your browser and visit http://localhost:8000/polls/, you shall see this upload form:



Choose an excel sheet, for example an xls file, and press "Submit". You will get a csv file for download.



Please open the file polls/views.py and focus on the following code section:

```
# Create your views here.
def upload(request):
    if request.method == "POST":
        form = UploadFileForm(request.POST, request.FILES)
        if form.is_valid():
            filehandle = request.FILES["file"]
            return excel.make_response(
                filehandle.get_sheet(), "csv", file_name="download"
            )
    else:
        form = UploadFileForm()
    return render(
        request,
        "upload_form.html",
        {
            "form": form,
            "title": "Excel file upload and download example",
            "header": (
                "Please choose any excel file "
                + "from your cloned repository:"
            ),
        },
```

UploadFileForm is html widget for file upload form in the html page. Then look down at **filehandle**. It is an instance of either ExcelInMemoryUploadedFile or TemporaryUploadedExcelFile, which inherit ExcelMixin and hence have a list of conversion methods to call, such as get_sheet, get_array, etc.

For the response, *`make_response()`* converts `pyexcel.Sheet` instance obtained via *`get_sheet()`* into a csv file for download.

Please feel free to change those functions according to *the mapping table*.

7.2 Handle data import

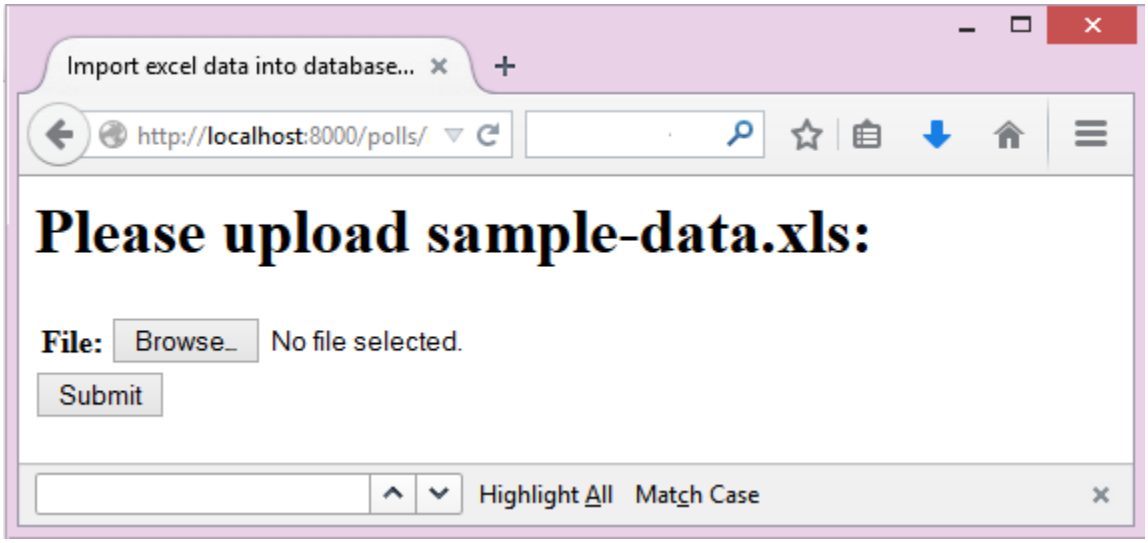
This example shows how to import uploaded excel file into django models. We are going to import sample-data.xls into the following data models:

```
class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    slug = models.CharField(max_length=10, unique=True,
                            default="question")

    def __str__(self):
        return self.question_text


class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)

    def __str__(self):
        return self.choice_text
```

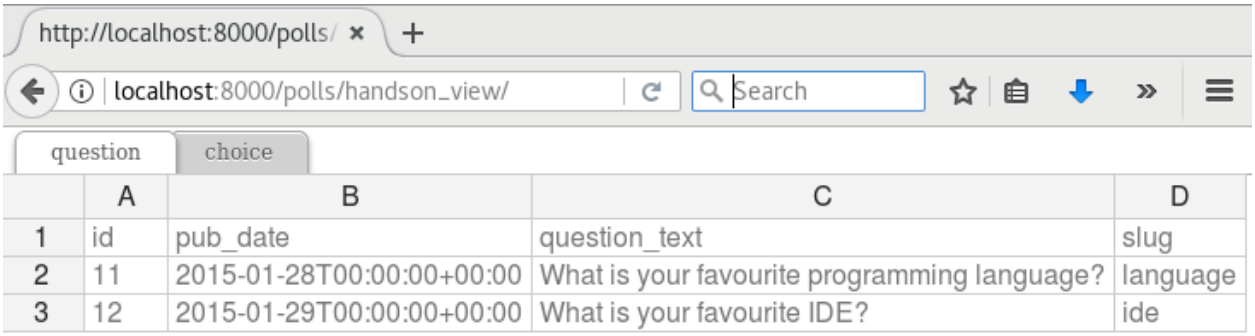
Note: Except the added "slug" field, **Question** and **Choice** are copied from Django tutorial part 1.

Note: Please also pay attention to 'choice' sheet. There exists an arbitrary column: "Noise", which exists to show case skipping column feature using mapdicts later.

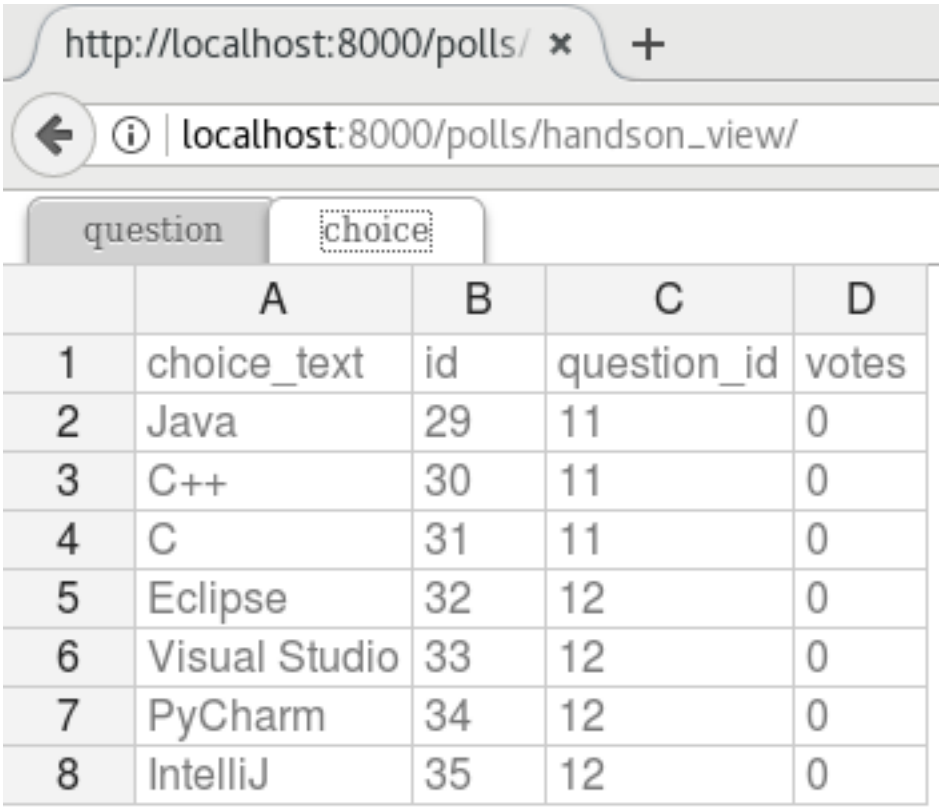
Please visit this link http://localhost:8000/polls/import/, you shall see this upload form:



Please then select sample-data.xls and upload. And you get the following excel-alike table in response to confirm all were imported.



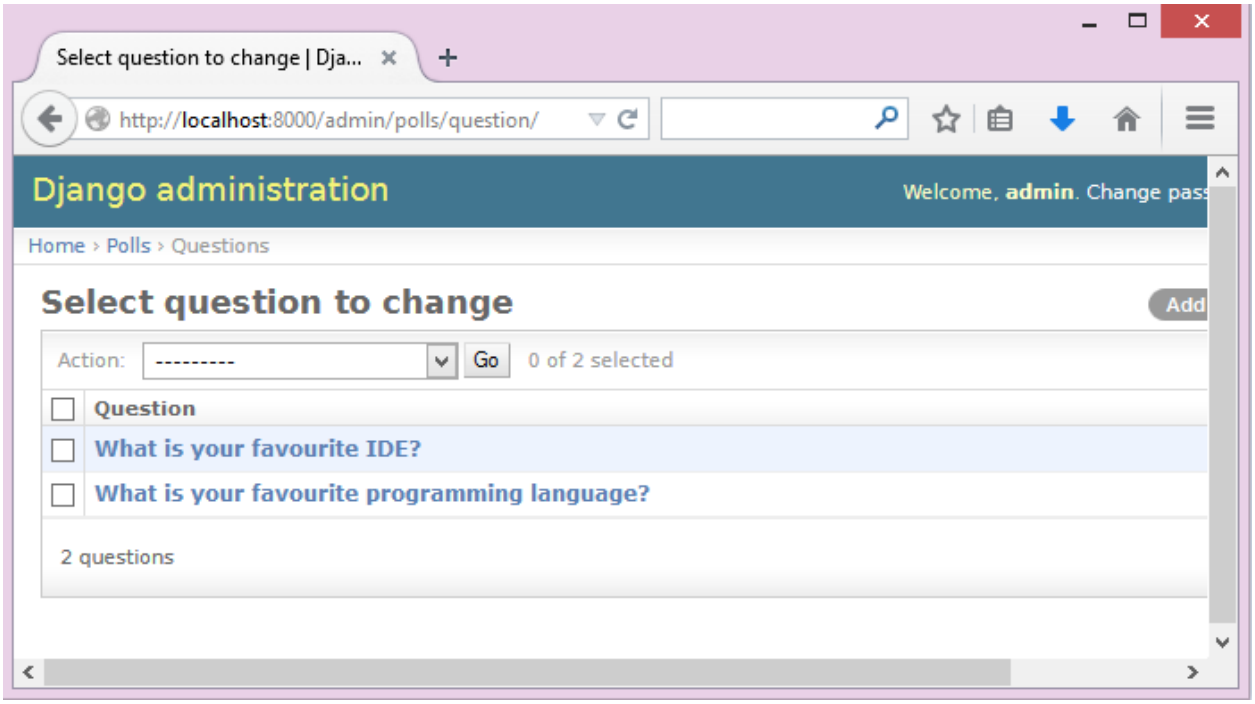
	A	B	C	D
1	id	pub_date	question_text	slug
2	11	2015-01-28T00:00:00+00:00	What is your favourite programming language?	language
3	12	2015-01-29T00:00:00+00:00	What is your favourite IDE?	ide



	A	B	C	D
1	choice_text	id	question_id	votes
2	Java	29	11	0
3	C++	30	11	0
4	C	31	11	0
5	Eclipse	32	12	0
6	Visual Studio	33	12	0
7	PyCharm	34	12	0
8	IntelliJ	35	12	0

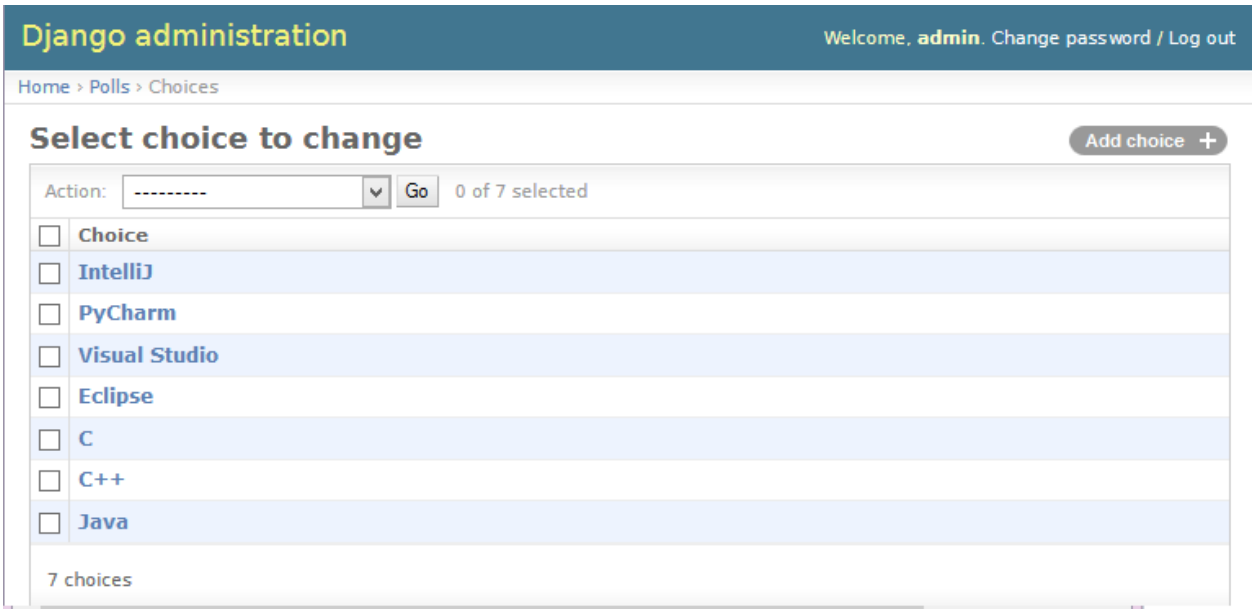
Note: pyexcel-handsontable along with pyexcel v0.5.0 brings excel-alie table rendering feature. Let me explain how this view is done a few paragraphs later.

Then visit the admin page http://localhost:8000/admin/polls/question, you shall see questions have been populated:



Note: The admin user credentials are: user name: admin, password: admin

And choices too:



You may use admin interface to delete all those objects and try again.

Now please open polls/views.py and focus on this part of code:

```
def import_data(request):
    if request.method == "POST":
        form = UploadFileForm(request.POST, request.FILES)

        def choice_func(row):
            q = Question.objects.filter(slug=row[0])[0]
            row[0] = q
            return row

        if form.is_valid():
            request.FILES["file"].save_book_to_database(
                models=[Question, Choice],
                initializers=[None, choice_func],
                mapdicts=[
                    ["question_text", "pub_date", "slug"],
                    {"Question": "question", "Choice": "choice_text", "Votes": "votes
↪"},
                ],
            )
            return redirect("handson_view")
        else:
            return HttpResponseBadRequest()
```

The star is `save_book_to_database()`. The parameter **models** should be a list of django models. **initializers** is a list of initialization functions for each model. In the example, we do not have init function for Question so 'None' is given and *choice_func* is given to Choice. **mapdicts** is a list of column names for each model and the member of the **mapdicts** can be a dictionary:

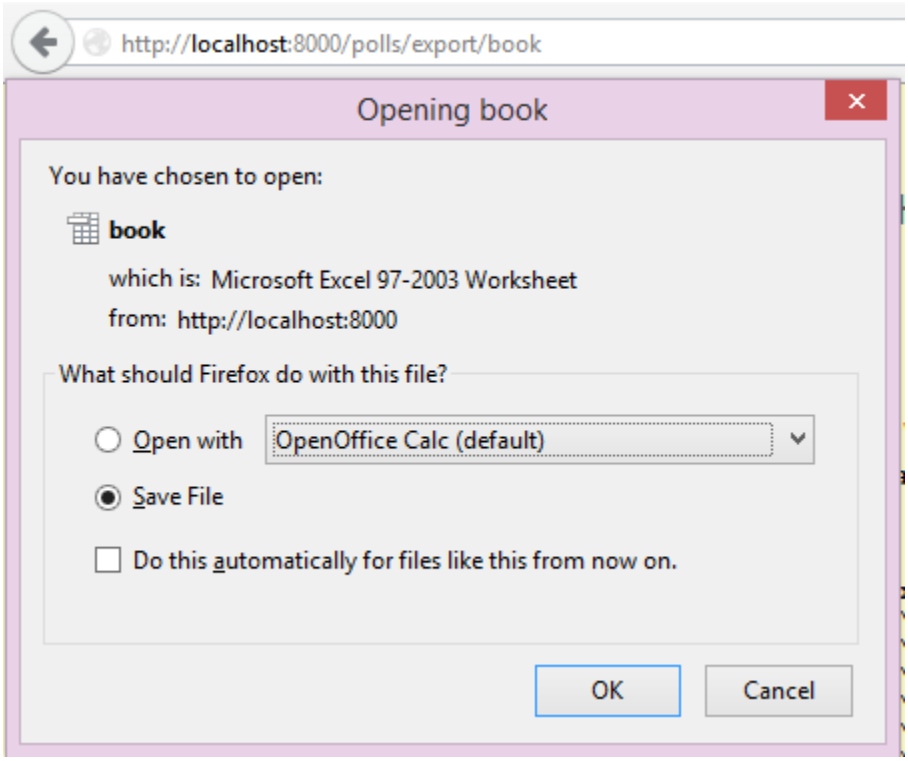
```
{
  "Question Text": "question_text",
  "Publish Date": "pub_date",
  "Unique Identifier": "slug"
}
```

As a dictionary, it can be used to skip columns in the incoming sheets. For example, 'Noise' column is skipped because it was not mentioned in the mapdict.

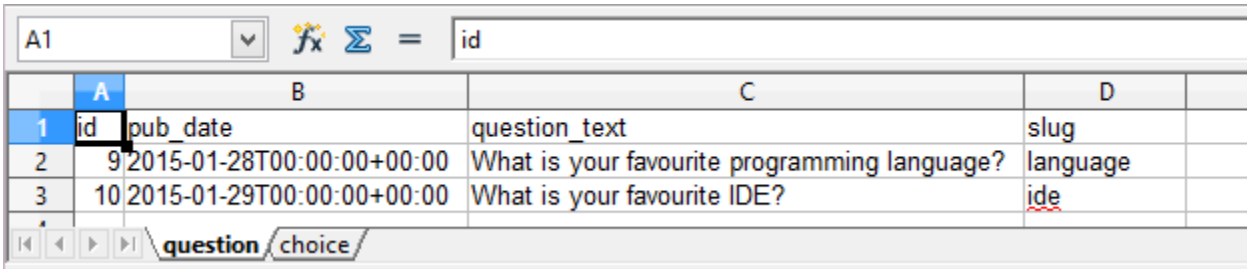
The custom initialization function is needed when the data from the excel sheet needs translation before data import. For example, **Choice** has a foreign key to **Question**. When choice data are to be imported, "Question" column needs to be translated to a question instance. In our example, "Question" column in "Sheet 2" contains the values appeared in "Unique Identifier" column in "Sheet 1".

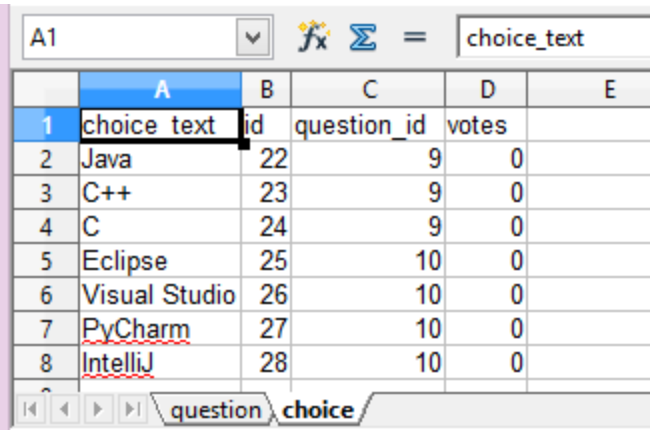
7.3 Handle data export

This section shows how to export the data in your models as an excel file. After you have completed the previous section, you can visit http://localhost:8000/polls/export/book and you shall get a file download dialog:



Please save and open it. You shall see these data in your window:





	A	B	C	D	E
1	choice_text	id	question_id	votes	
2	Java	22	9	0	
3	C++	23	9	0	
4	C	24	9	0	
5	Eclipse	25	10	0	
6	Visual Studio	26	10	0	
7	PyCharm	27	10	0	
8	IntelliJ	28	10	0	

Now let's examine the code behind this in polls/views.py:

```
def export_data(request, atype):
    if atype == "sheet":
        return excel.make_response_from_a_table(
            Question, "xls", file_name="sheet"
        )
    elif atype == "book":
        return excel.make_response_from_tables(
            [Question, Choice], "xls", file_name="book"
```

make_response_from_tables() does all what is needed: read out the data, convert them into xls and give it the browser. And what you need to do is to give a list of models to be exported and a file type. As you have noticed, you can visit http://localhost:8000/polls/export/sheet and will get **Question** exported as a single sheet file.

7.4 Render an excel-alike html in a browser

In previous section, you have seen the rendering of the excel-alike table. First of all, the credits goes to handsontable developers. pyexcel-handsontable as renderer plugin to pyexcel v0.5.0 bring it to pyexcel developers.

Here is how it is done. Simply put in 'handsontable.html' instead of 'xls' as file type.

```
    return excel.make_response_from_tables(
        [Question, Choice], "handsontable.html"
    )
```

It is understood that you will want to embed it into your django templates. Here are the sample embedding code:

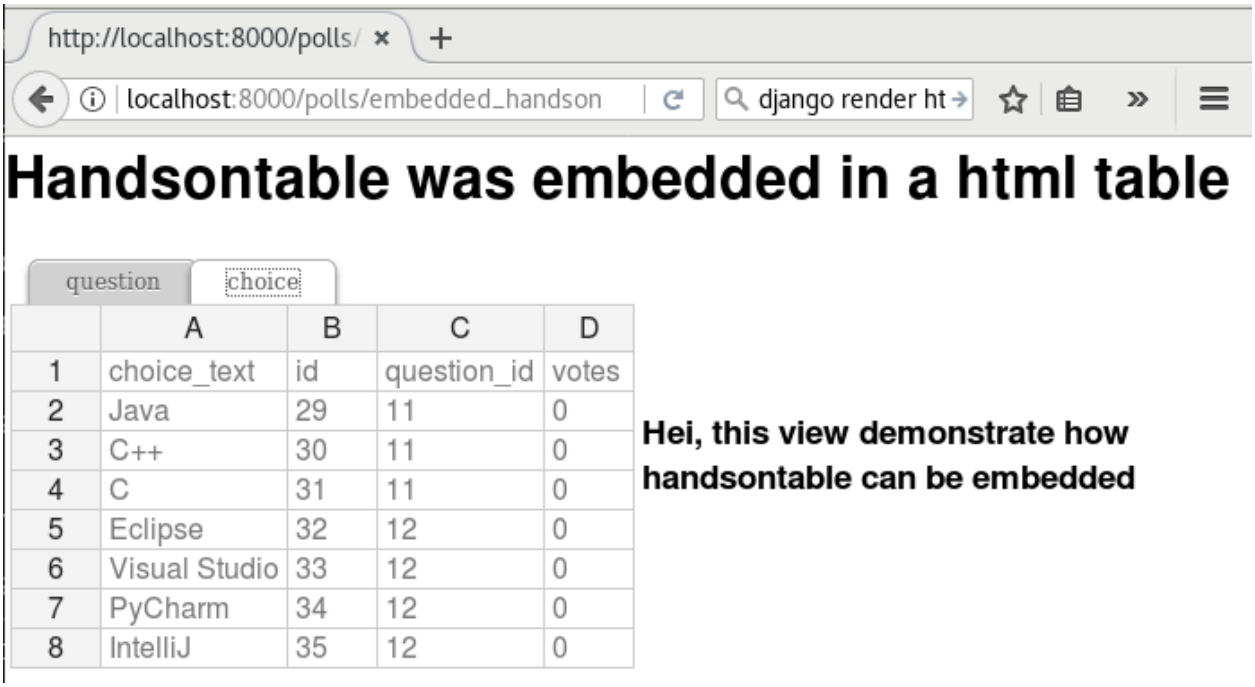
```
def embed_handson_table(request):
    """
    Renders two table in a handsontable
    """
    content = excel.pe.save_book_as(
        models=[Question, Choice],
        dest_file_type="handsontable.html",
        dest_embed=True,
    )
    content.seek(0)
    return render(
        request,
        "custom-handson-table.html",
```

(continues on next page)

(continued from previous page)

```
        {"handsontable_content": content.read()},
    )


def embed_handson_table_from_a_single_table(request):
    """
    Renders one table in a handsontable
    """
    content = excel.pe.save_as(
        model=Question, dest_file_type="handsontable.html", dest_embed=True
    )
    content.seek(0)
    return render(
        request,
        "custom-handson-table.html",
        {"handsontable_content": content.read()},
    )
```

Those views can be accessed as http://localhost:8000/polls/embedded_handson_view/ and http://localhost:8000/polls/embedded_handson_view_single/.



	A	B	C	D
1	choice_text	id	question_id	votes
2	Java	29	11	0
3	C++	30	11	0
4	C	31	11	0
5	Eclipse	32	12	0
6	Visual Studio	33	12	0
7	PyCharm	34	12	0
8	IntelliJ	35	12	0

7.4.1 How to import one sheet instead of multi-sheet book

Previous example shows how to import a multi-sheet book. However, what exactly is needed to import only one sheet instead? Before you proceed, please empty question and choice data using django admin.

Let's visit this url first http://localhost:8000/polls/imports_sheet/, where you see a similar file upload form. This time please choose sample-sheet.xls instead. Then look at django admin and see if the question data have been imported or not.

Now let's look at the code:

```
def import_sheet(request):
    if request.method == "POST":
        form = UploadFileForm(request.POST, request.FILES)
        if form.is_valid():
            request.FILES["file"].save_to_database(
                name_columns_by_row=2,
                model=Question,
                mapdict=["question_text", "pub_date", "slug"],
            )
            return HttpResponse("OK")
        else:
```

Because it is a single sheet, the function to call is *save_to_database()* where you specify a model and its mapping dictionary.

Have you noticed the extra parameter 'name_columns_by_row'? Why is this needed? Well, normally you *will not need* that if you have column names in the first row. In this example, the column names appears in the second row. Please open sample-sheet.xls and have a look. The straight answer is because the column names in the data appears in the 2nd row of the data matrix.

Note: If you have imported earlier excel sheet "sample-data.xls", you will get the following warning in your console output:

```
Warning: Bulk insertion got below exception. Trying to do it one by one slowly.
column slug is not unique <- reason
One row is ignored <- action
column slug is not unique
What is your favourite programming language?
One row is ignored
column slug is not unique
What is your favourite IDE?
```

This is because question data have been imported before. Django is raising IntegrityError. For more details please read this part of code in pyexcel-io, and django-excel issue 2

In order to remove those warnings, what you can do is to empty all data using django admin and redo this single sheet import again.

7.4.2 What to do if import data overlaps existing data in the database

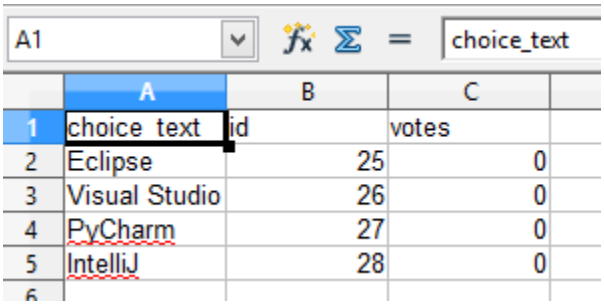
With new version pyexcel-io v0.1.0, you could provide the row initialization function that returns None in order to skip a row in your import data. Inside the initialization function, you could also do database update. As long as it returns None, django-excel will try to do bulk create the import data.

7.5 Handle custom data export

It is also quite common to download a portion of the data in a database table, for example the result of a search query. With version 0.0.2, you can pass on a query sets to to *make_response_from_query_sets()* and generate an excel sheet from it:

```
def export_data(request, atype):
            [Question, Choice], "xls", file_name="book"
        )
    elif atype == "custom":
        question = Question.objects.get(slug="ide")
        query_sets = Choice.objects.filter(question=question)
        column_names = ["choice_text", "id", "votes"]
        return excel.make_response_from_query_sets(
            query_sets, column_names, "xls", file_name="custom"
        )
    else:
```

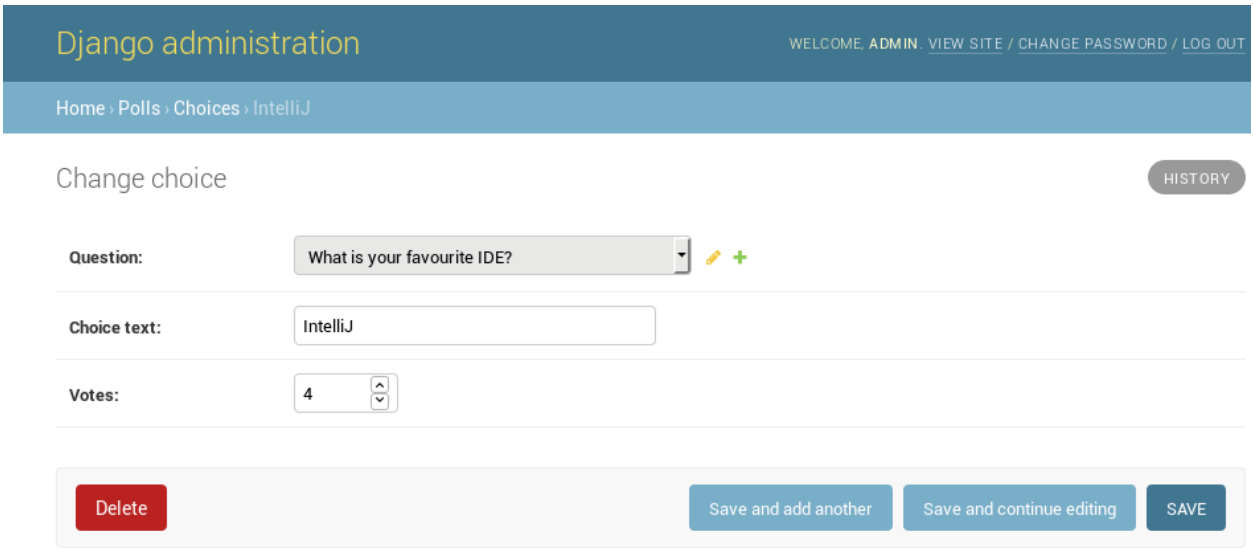
You can visit http://localhost:8000/polls/export/custom and will get the query set exported as a single sheet file as:



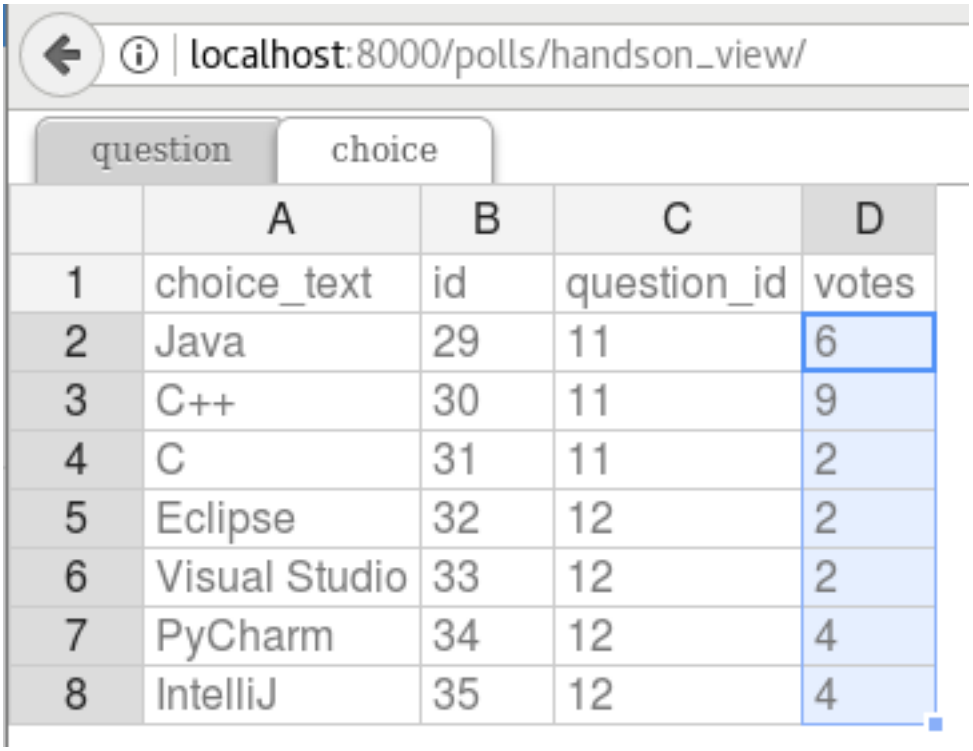
	A	B	C
1	choice_text	id	votes
2	Eclipse	25	0
3	Visual Studio	26	0
4	PyCharm	27	0
5	IntelliJ	28	0

7.6 Visualize your data

Let's go to the admin page and update some votes for the choices.

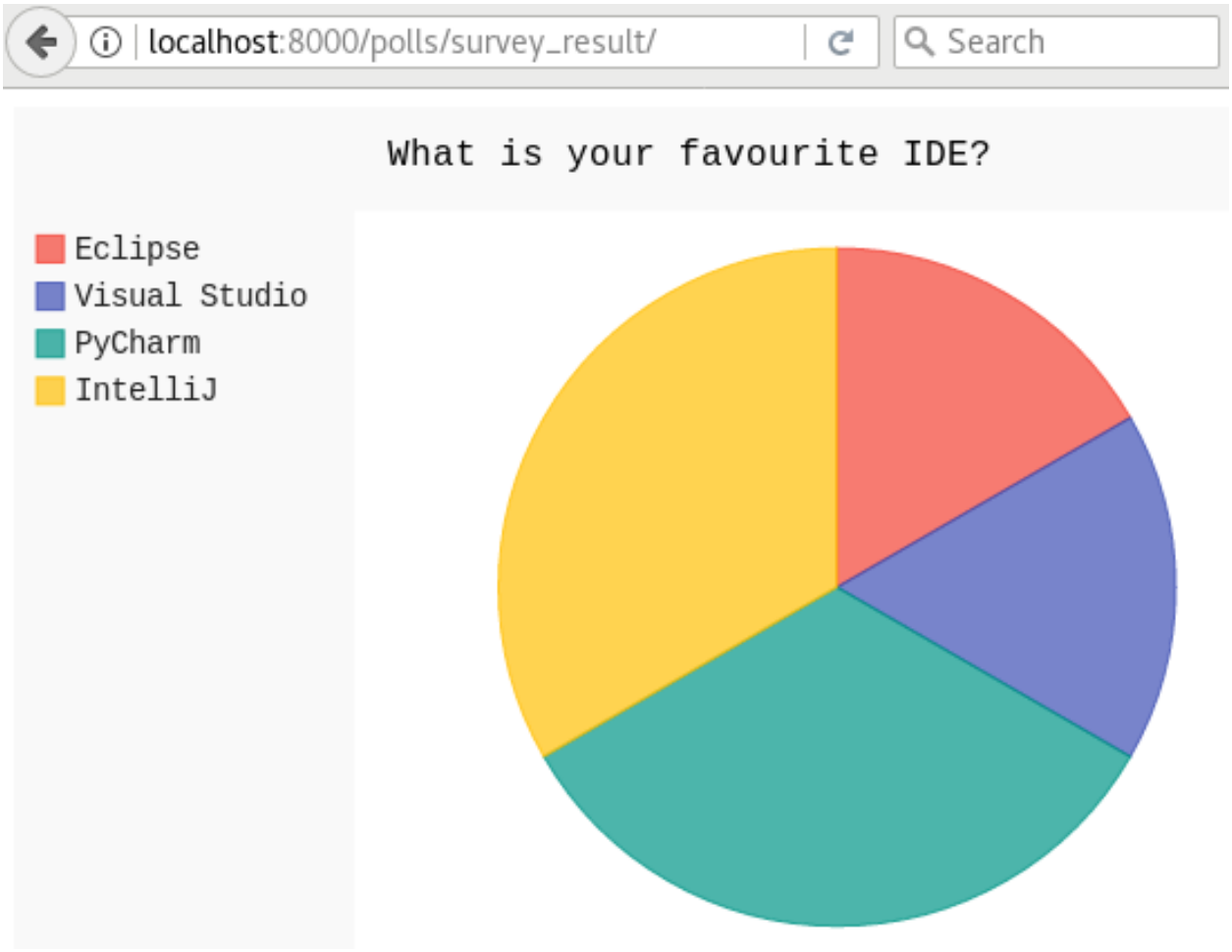


In my case, I have updated all of them and have gotten something like this:



	A	B	C	D
1	choice_text	id	question_id	votes
2	Java	29	11	6
3	C++	30	11	9
4	C	31	11	2
5	Eclipse	32	12	2
6	Visual Studio	33	12	2
7	PyCharm	34	12	4
8	IntelliJ	35	12	4

Now, let's look at the survey result(http://localhost:8000/polls/survey_result/) for "What's your favorite IDE?":



pyexcel-pygal provide you the common data visualization capability to show your data intuitively. Here is the code to achieve that:

```
    query_sets = Choice.objects.filter(question=question)
    column_names = ["choice_text", "votes"]

    # Obtain a pyexcel sheet from the query sets
    sheet = excel.pe.get_sheet(
        query_sets=query_sets, column_names=column_names
    )
    sheet.name_columns_by_row(0)
    sheet.column.format("votes", int)

    # Transform the sheet into an svg chart
    svg = excel.pe.save_as(
        array=[sheet.column["choice_text"], sheet.column["votes"]],
        dest_file_type="svg",
        dest_chart_type="pie",
        dest_title=question.question_text,
        dest_width=600,
        dest_height=400,
    )

    return render(request, "survey_result.html", dict(svg=svg.read()))
```

(continues on next page)

(continued from previous page)

```
def import_sheet_using_isave_to_database(request):
    if request.method == "POST":
        form = UploadFileForm(request.POST, request.FILES)
```

CHAPTER 8

All supported data types

The example application likes to have array but it is not just about arrays. Here is table of functions for all supported data types:

data structure	from file to data structures	from data structures to response
dict	*get_dict()*	*make_response_from_dict()*
records	*get_records()*	*make_response_from_records()*
a list of lists	*get_array()*	*make_response_from_array()*
dict of a list of lists	*get_book_dict()*	*make_response_from_book_dict()*
pyexcel.Sheet	*get_sheet()*	*make_response()*
pyexcel.Book	*get_book()*	*make_response()*
database table	*save_to_database()* *isave_to_database()*	*make_response_from_a_table()*
a list of database tables	*save_book_to_database()* *isave_book_to_database()*	*make_response_from_tables()*
a database query sets		*make_response_from_query_sets()*
a generator for records	*iget_records()*	
a generator of lists	*iget_array()*	

See more examples of the data structures in pyexcel documentation

CHAPTER 9

API Reference

django-excel attaches **pyexcel** functions to **InMemoryUploadedFile** and **TemporaryUploadedFile**. Hence, the following functions are available for the uploaded files, e.g. request.FILES['your_uploaded_file'].

`django_excel.ExcelMixin.`**`get_sheet`**(*sheet_name=None*, ***keywords*)

Parameters

- **`sheet_name`** – For an excel book, there could be multiple sheets. If it is left unspecified, the sheet at index 0 is loaded. For 'csv', 'tsv' file, *sheet_name* should be None anyway.
- **`keywords`** – additional keywords to `pyexcel.get_sheet()`

Returns A sheet object

`django_excel.ExcelMixin.`**`get_array`**(*sheet_name=None*, ***keywords*)

Parameters

- **`sheet_name`** – same as *`get_sheet()`*
- **`keywords`** – additional keywords to `pyexcel.get_array()`

Returns a two dimensional array, a list of lists

`django_excel.ExcelMixin.`**`iget_array`**(*sheet_name=None*, ***keywords*)

Parameters

- **`sheet_name`** – same as *`get_sheet()`*
- **`keywords`** – additional keywords to `pyexcel.iget_array()`

Returns a generator for a two dimensional array, a list of lists

`django_excel.ExcelMixin.`**`get_dict`**(*sheet_name=None*, *name_columns_by_row=0*, ***keywords*)

Parameters

- **`sheet_name`** – same as *`get_sheet()`*
- **`name_columns_by_row`** – uses the first row of the sheet to be column headers by default.

- **keywords** – additional keywords to `pyexcel.get_dict()`

Returns a dictionary of the file content

`django_excel.ExcelMixin.`**`get_records`**(*sheet_name=None*, *name_columns_by_row=0*, ***keywords*)

Parameters

- **sheet_name** – same as *`get_sheet()`*
- **name_columns_by_row** – uses the first row of the sheet to be record field names by default.
- **keywords** – additional keywords to `pyexcel.get_records()`

Returns a list of dictionary of the file content

`django_excel.ExcelMixin.`**`iget_records`**(*sheet_name=None*, *name_columns_by_row=0*, ***keywords*)

Parameters

- **sheet_name** – same as *`get_sheet()`*
- **name_columns_by_row** – uses the first row of the sheet to be record field names by default.
- **keywords** – additional keywords to `pyexcel.iget_records()`

Returns a generator for a list of dictionary of the file content

`django_excel.ExcelMixin.`**`get_book`**(***keywords*)

Parameters **keywords** – additional keywords to `pyexcel.get_book()`

Returns a two dimensional array, a list of lists

`django_excel.ExcelMixin.`**`get_book_dict`**(***keywords*)

Parameters **keywords** – additional keywords to `pyexcel.get_book_dict()`

Returns a two dimensional array, a list of lists

`django_excel.ExcelMixin.`**`save_to_database`**(*model=None*, *initializer=None*, *mapdict=None*, ***keywords*)

Parameters

- **model** – a django model
- **initializer** – a custom table initialization function if you have one
- **mapdict** – the explicit table column names if your excel data do not have the exact column names
- **keywords** – additional keywords to `pyexcel.Sheet.save_to_django_model()`

`django_excel.ExcelMixin.`**`isave_to_database`**(*model=None*, *initializer=None*, *mapdict=None*, ***keywords*)

similar to *`save_to_database()`*. But it requires less memory.

This requires column names must be at the first row.

`django_excel.ExcelMixin.`**`save_book_to_database`**(*models=None*, *initializers=None*, *mapdicts=None*, ***keywords*)

Parameters

- **models** – a list of django models

- **initializers** – a list of model initialization functions.
- **mapdicts** – a list of explicit table column names if your excel data sheets do not have the exact column names
- **keywords** – additional keywords to `pyexcel.Book.save_to_django_models()`

`django_excel.ExcelMixin.`**`isave_book_to_database`**(*models=None*, *initializers=None*, *mapdicts=None*, ***keywords*)

similar to *`save_book_to_database()`*. But it requires less memory.

This requires column names must be at the first row in each sheets.

`django_excel.ExcelMixin.`**`free_resources`**()

It should be called after iget_array and iget_records were used

CHAPTER 10

Response methods

django_excel.**make_response**(*pyexcel_instance*, *file_type*, *status=200*)

Parameters

- **pyexcel_instance** – pyexcel.Sheet or pyexcel.Book
- **file_type** – one of the following strings:
  - 'csv'
  - 'tsv'
  - 'csvz'
  - 'tsvz'
  - 'xls'
  - 'xlsx'
  - 'xlsm'
  - 'ods'
- **status** – unless a different status is to be returned.

django_excel.**make_response_from_array**(*array*, *file_type*, *status=200*)

Parameters

- **array** – a list of lists
- **file_type** – same as *make_response()*
- **status** – same as *make_response()*

django_excel.**make_response_from_dict**(*dict*, *file_type*, *status=200*)

Parameters

- **dict** – a dictionary of lists
- **file_type** – same as *make_response()*

- **status** – same as *make_response()*

django_excel.**make_response_from_records**(*records*, *file_type*, *status=200*)

Parameters

- **records** – a list of dictionaries
- **file_type** – same as *make_response()*
- **status** – same as *make_response()*

django_excel.**make_response_from_book_dict**(*book_dict*, *file_type*, *status=200*)

Parameters

- **book_dict** – a dictionary of two dimensional arrays
- **file_type** – same as *make_response()*
- **status** – same as *make_response()*

django_excel.**make_response_from_a_table**(*model*, *file_type status=200*)
Produce a single sheet Excel book of *file_type*

Parameters

- **model** – a Django model
- **file_type** – same as *make_response()*
- **status** – same as *make_response()*

django_excel.**make_response_from_query_sets**(*query_sets*, *column_names*, *file_type status=200*)

Produce a single sheet Excel book of *file_type* from your custom database queries

Parameters

- **query_sets** – a query set
- **column_names** – a nominated column names. It could not be None, otherwise no data is returned.
- **file_type** – same as *make_response()*
- **status** – same as *make_response()*

django_excel.**make_response_from_tables**(*models*, *file_type status=200*)

Produce a multiple sheet Excel book of *file_type*. It becomes the same as *make_response_from_a_table()* if you pass *tables* with an array that has a single table

Parameters

- **models** – a list of Django models
- **file_type** – same as *make_response()*
- **status** – same as *make_response()*

Python Module Index

d

Index