Contents

[Problem: 3](#_Toc498095197)

[Objective: 3](#_Toc498095198)

[1. PRE-REQUISITES: 3](#_Toc498095199)

[2. SETUP 6](#_Toc498095200)

[a. Create virtual env 6](#_Toc498095201)

[b. Install pre-requisites 6](#_Toc498095202)

[c. Create Django project 7](#_Toc498095203)

[d. Create Django app 7](#_Toc498095204)

[Ilust1.Initial folder structure 8](#_Toc498095205)

[e. Add app to settings.py 8](#_Toc498095206)

[f. Create super user (admin/admin123) 8](#_Toc498095207)

[g. Start the server and check the site so far 9](#_Toc498095208)

[3. MODEL 9](#_Toc498095209)

[a. Create model 9](#_Toc498095210)

[b. Update admin.py with the new model 9](#_Toc498095211)

[c. Add import\_export 9](#_Toc498095212)

[d. Update settings.py for import\_export module 9](#_Toc498095213)

[e. Define Resource/ModelResource: 9](#_Toc498095214)

[f. Update admin.py 10](#_Toc498095215)

[4. VIEWS.PY and URLS.PY – CREATE A NEW TEST PAGE 10](#_Toc498095216)

[a. Updated tbc\_sri\_app/views.py 10](#_Toc498095217)

[b. Created tbc\_sri\_app/urls.py 11](#_Toc498095218)

[c. Updated tbc\_sri/urls.py 11](#_Toc498095219)

[5. GRID INTRO 11](#_Toc498095220)

[a. Grid tool to use 11](#_Toc498095221)

[b. Download jsgrid package 12](#_Toc498095222)

[Ilust2. New static folder with JSGrid files 12](#_Toc498095223)

[c. Check settings.py 12](#_Toc498095224)

[d. Create a test page 13](#_Toc498095225)

[Ilust3. Templates folder with index.html 13](#_Toc498095226)

[e. Update urls.py and views.py with the test pages 13](#_Toc498095227)

[6. CONNECTING THE MODEL AND THE GRID 14](#_Toc498095228)

[a. loadData 14](#_Toc498095229)

[b. updateItem 17](#_Toc498095230)

[Branching to RESTful01 21](#_Toc498095231)

[References 23](#_Toc498095232)

[1. Django 1.11 official Tutorial 23](#_Toc498095233)

[2. Django Import\_Export 23](#_Toc498095234)

[3. How to serve static files including collectstatics 23](#_Toc498095235)

[4. JSGrid 23](#_Toc498095236)

[5. JQGrid 23](#_Toc498095237)

**TBC\_SRI**

# Problem:

Daily SRI report is taking up 5 person hours daily. The data is stored as a spreadsheet and shared via email/sharepoint/public folder. Compiled report is emailed to TBC/FDSI daily. Not clear if the 315 data is incorporated into the report but suspecting they are not. There is now an idea to download the Pipeline report from GLN, create an Excel macro to generate a pseudo “SRI Report” which will then be updated by ABA6 operators and then uploaded back into the GLN. This data will then be sent to FDSI as 315. If it works, the plan is to negotiate with FDSI to terminate SRI report.

1. Because it is macro based, it is still managing data on a spread sheet than a central DB, i.e., multiple version of the “fact”.
2. Being a spot solution, it will be hard to integrate with any other tasks to create a “flow”.

# Objective:

Create a DB that will hold the tracking data at the container level. Should be the central repository for the data where operators can view and update data.

Reports can be generated to share with customers on cargo movement status.

Operators can use this system to look up the status of the cargoes.

Should be manually editable on the table.

Also able to mass upload/update data via CSV file.

<https://www.ibm.com/developerworks/library/wa-django/>

# PRE-REQUISITES:

* Python 2.5+ : I have 3.6.1
* Simplejson:
* Django 1.2.3: 1.11.1 installed

**Install simplejson**

*C:\Users\msugimoto>****pip install simplejson***

*Collecting simplejson*

*Downloading simplejson-3.11.1.tar.gz (78kB)*

*100% |████████████████████████████████| 81kB 750kB/s*

*Installing collected packages: simplejson*

*Running setup.py install for simplejson ... done*

*Successfully installed simplejson-3.11.1*

**Install jquery**

*C:\Users\msugimoto>****pip install django-jquery***

*Collecting django-jquery*

*Downloading django\_jquery-3.1.0-py2.py3-none-any.whl*

*Requirement already satisfied: Django>=1.3 in c:\users\msugimoto\appdata\local\programs\python\python36\lib\site-packages (from django-jquery)*

*Requirement already satisfied: pytz in c:\users\msugimoto\appdata\local\programs\python\python36\lib\site-packages (from Django>=1.3->django-jquery)*

*Installing collected packages: django-jquery*

*Successfully installed django-jquery-3.1.0*

**Install jquery ui**

*C:\Users\msugimoto>****pip install django-jquery-ui***

*Collecting django-jquery-ui*

*Downloading django-jquery-ui-1.11.4.1.tar.gz (1.2MB)*

*100% |████████████████████████████████| 1.2MB 729kB/s*

*Requirement already satisfied: django-jquery>=1.6 in c:\users\msugimoto\appdata\local\programs\python\python36\lib\site-packages (from django-jquery-ui)*

*Requirement already satisfied: Django>=1.3 in c:\users\msugimoto\appdata\local\programs\python\python36\lib\site-packages (from django-jquery>=1.6->django-jquer*

*y-ui)*

*Requirement already satisfied: pytz in c:\users\msugimoto\appdata\local\programs\python\python36\lib\site-packages (from Django>=1.3->django-jquery>=1.6->django*

*-jquery-ui)*

*Installing collected packages: django-jquery-ui*

*Running setup.py install for django-jquery-ui ... done*

*Successfully installed django-jquery-ui-1.11.4.1*

Review (comments) are pretty bad so decided to shift gears and try out jsgrid instead.

<https://github.com/tabalinas/jsgrid#requirement>

Specifically, I went to the Django sub at

<https://github.com/tabalinas/jsgrid-django>

created a requirements.txt file and copy/pasted the contents.

Then following instructions ran the pip install

*C:\DjangoLab\TBC\_SRI\jsgrid>****pip install -r requirements.txt***

*Collecting certifi==2017.4.17 (from -r requirements.txt (line 1))*

*Using cached certifi-2017.4.17-py2.py3-none-any.whl*

*Collecting chardet==3.0.4 (from -r requirements.txt (line 2))*

*Downloading chardet-3.0.4-py2.py3-none-any.whl (133kB)*

*100% |████████████████████████████████| 143kB 1.3MB/s*

*Collecting Django==1.8 (from -r requirements.txt (line 3))*

*Downloading Django-1.8-py2.py3-none-any.whl (6.2MB)*

*100% |████████████████████████████████| 6.2MB 211kB/s*

*Collecting django-simple-rest==1.4.1 (from -r requirements.txt (line 4))*

*Downloading django-simple-rest-1.4.1.tar.gz*

*Collecting idna==2.5 (from -r requirements.txt (line 5))*

*Using cached idna-2.5-py2.py3-none-any.whl*

*Collecting mimeparse==0.1.3 (from -r requirements.txt (line 6))*

*Downloading mimeparse-0.1.3.tar.gz*

*Collecting requests==2.17.3 (from -r requirements.txt (line 7))*

*Using cached requests-2.17.3-py2.py3-none-any.whl*

*Collecting urllib3==1.21.1 (from -r requirements.txt (line 8))*

*Using cached urllib3-1.21.1-py2.py3-none-any.whl*

*Collecting wheel==0.24.0 (from -r requirements.txt (line 9))*

*Downloading wheel-0.24.0-py2.py3-none-any.whl (63kB)*

*100% |████████████████████████████████| 71kB 2.3MB/s*

*Requirement already satisfied: setuptools in c:\users\msugimoto\appdata\local\programs\python\python36\lib\site-packages (from django-simple-rest==1.4.1->-r req*

*uirements.txt (line 4))*

*Installing collected packages: certifi, chardet, Django, mimeparse, django-simple-rest, idna, urllib3, requests, wheel*

*Found existing installation: Django 1.11.1*

*Uninstalling Django-1.11.1:*

*Successfully uninstalled Django-1.11.1*

*Running setup.py install for mimeparse ... done*

*Running setup.py install for django-simple-rest ... done*

*Successfully installed Django-1.8 certifi-2017.4.17 chardet-3.0.4 django-simple-rest-1.4.1 idna-2.5 mimeparse-0.1.3 requests-2.17.3 urllib3-1.21.1 wheel-0.24.0*

# SETUP

## Create virtual env

*C:\DjangoLab\TBC\_SRI>****virtualenv tbc\_sri\_env***

*Using base prefix 'c:\\users\\msugimoto\\appdata\\local\\programs\\python\\python36'*

*New python executable in C:\DjangoLab\TBC\_SRI\tbc\_sri\_env\Scripts\python.exe*

*Installing setuptools, pip, wheel...done.*

*C:\DjangoLab\TBC\_SRI>****cd tbc\_sri\_env***

*C:\DjangoLab\TBC\_SRI\tbc\_sri\_env>****scripts\activate.bat***

## Install pre-requisites

Will reuse the requirements.txt from step 0.

*(tbc\_sri\_env) C:\DjangoLab\TBC\_SRI\tbc\_sri\_env>****cd C:\DjangoLab\TBC\_SRI\jsgrid***

*(tbc\_sri\_env) C:\DjangoLab\TBC\_SRI\jsgrid>****pip install -r requirements.txt***

*Collecting certifi==2017.4.17 (from -r requirements.txt (line 1))*

*Using cached certifi-2017.4.17-py2.py3-none-any.whl*

*Collecting chardet==3.0.4 (from -r requirements.txt (line 2))*

*Using cached chardet-3.0.4-py2.py3-none-any.whl*

*Collecting Django==1.8 (from -r requirements.txt (line 3))*

*Using cached Django-1.8-py2.py3-none-any.whl*

*Collecting django-simple-rest==1.4.1 (from -r requirements.txt (line 4))*

*Using cached django-simple-rest-1.4.1.tar.gz*

*Collecting idna==2.5 (from -r requirements.txt (line 5))*

*Using cached idna-2.5-py2.py3-none-any.whl*

*Collecting mimeparse==0.1.3 (from -r requirements.txt (line 6))*

*Using cached mimeparse-0.1.3.tar.gz*

*Collecting requests==2.17.3 (from -r requirements.txt (line 7))*

*Using cached requests-2.17.3-py2.py3-none-any.whl*

*Collecting urllib3==1.21.1 (from -r requirements.txt (line 8))*

*Using cached urllib3-1.21.1-py2.py3-none-any.whl*

*Collecting wheel==0.24.0 (from -r requirements.txt (line 9))*

*Using cached wheel-0.24.0-py2.py3-none-any.whl*

*Requirement already satisfied: setuptools in c:\djangolab\tbc\_sri\tbc\_sri\_env\lib\site-packages (from django-simple-rest==1.4.1->-r requirements.txt (line 4))*

*Building wheels for collected packages: django-simple-rest, mimeparse*

*Running setup.py bdist\_wheel for django-simple-rest ... done*

*Stored in directory: C:\Users\msugimoto\AppData\Local\pip\Cache\wheels\41\ba\c3\5a979a30223fad2e1236ccdceedda59d7eff9ab38a6b86b5b2*

*Running setup.py bdist\_wheel for mimeparse ... done*

*Stored in directory: C:\Users\msugimoto\AppData\Local\pip\Cache\wheels\cb\83\03\ec75acce6afbbecd7aaf161c59554eb64c04fc3bfdeb117a44*

*Successfully built django-simple-rest mimeparse*

*Installing collected packages: certifi, chardet, Django, mimeparse, django-simple-rest, idna, urllib3, requests, wheel*

*Found existing installation: Django 1.11.6*

*Uninstalling Django-1.11.6:*

*Successfully uninstalled Django-1.11.6*

*Found existing installation: wheel 0.30.0*

*Uninstalling wheel-0.30.0:*

*Successfully uninstalled wheel-0.30.0*

*Successfully installed Django-1.8 certifi-2017.4.17 chardet-3.0.4 django-simple-rest-1.4.1 idna-2.5 mimeparse-0.1.3 requests-2.17.3 urllib3-1.21.1 wheel-0.24.0*

## Create Django project

*(tbc\_sri\_env) C:\DjangoLab\TBC\_SRI\tbc\_sri\_env>****mkdir my\_source***

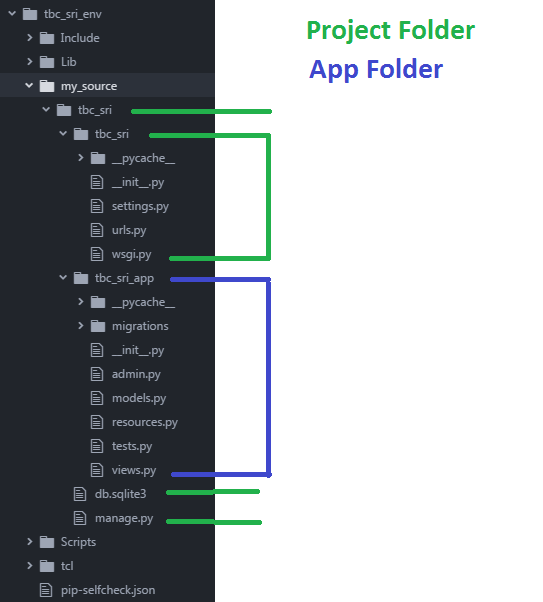
*(tbc\_sri\_env) C:\DjangoLab\TBC\_SRI\tbc\_sri\_env>****cd my\_source***

*(tbc\_sri\_env) C:\DjangoLab\TBC\_SRI\tbc\_sri\_env\my\_source>****django-admin.py startproject tbc\_sri***

## Create Django app

(tbc\_sri\_env) C:\DjangoLab\TBC\_SRI\tbc\_sri\_env\my\_source>***cd tbc\_sri***

(tbc\_sri\_env) C:\DjangoLab\TBC\_SRI\tbc\_sri\_env\my\_source\tbc\_sri>***django-admin.py startapp tbc\_sri\_app***

******

### Ilust1.Initial folder structure

## Add app to settings.py

INSTALLED\_APPS = (

…

***'tbc\_sri\_app',***

)

## Create super user (admin/admin123)

*(tbc\_sri\_env) C:\DjangoLab\TBC\_SRI\tbc\_sri\_env\my\_source\tbc\_sri>****python manage.py createsuperuser***

## Start the server and check the site so far

***python manage.py runserver***

<http://127.0.0.1:8000/admin>

So far so good…

# MODEL

## Create model

***python manage.py makemigrations***

***python manage.py migrate***

## Update admin.py with the new model

***from .models import lnos\_statusPipeLine***

***admin.site.register(lnos\_statusPipeLine)***

## Add import\_export

*(tbc\_sri\_env) C:\DjangoLab\TBC\_SRI\tbc\_sri\_env\Scripts>****pip install django\_import\_export***

## Update settings.py for import\_export module

*INSTALLED\_APPS = [*

*…*

*'import\_export',*

*]*

*…*

*IMPORT\_EXPORT\_USE\_TRANSACTIONS = True*

## Define Resource/ModelResource:

Create a “Resource” file, ***tbc\_sri\_app/resource.py***, which defines how objects are mapped to their import/export representations and handle importing/exporting data.

“ModelResource” on the other hand is a Resource subclass for handling Django models

(<http://django-import-export.readthedocs.io/en/latest/api_resources.html>)

***from import\_export import resources***

***from import\_export import fields***

***from import\_export.widgets import ForeignKeyWidget***

***from .models import lnos\_statusPipeLine***

***class Meta:***

***model = lnos\_statusPipeLine***

***skip\_unchanged = True***

***import\_id\_fields = (***

***…***

## Update admin.py

***from .resources import lnos\_statusPipeLineResource***

***from import\_export.admin import ImportExportModelAdmin***

***# Register your models here.***

***from .models import lnos\_statusPipeLine***

***@admin.register(lnos\_statusPipeLine)***

***class lnos\_statusPipeLineAdmin(ImportExportModelAdmin):***

***list\_display = ('mbol', 'hbol', 'container', 'customs\_released', 'eta\_pod', 'unloaded\_from\_vessel')***

***resource\_class = lnos\_statusPipeLineResource***

Try importing test data but got the error

***NOT NULL constraint failed: tbc\_sri\_app\_lnos\_statuspipeline.eta\_placeofdelivery***

All the CharFields were made ***null=True, blank=True*** but did not help.

Turns out I was forgetting to delete the old sql3 file.

# VIEWS.PY and URLS.PY – CREATE A NEW TEST PAGE

Will create a complete TBC shipment table first.

But first, noticed my Django version is 1.8 so need to upgrade to 1.11.6.

***Pip install -U django***

1. Updated tbc\_sri\_app/views.py with some Hello World pages

***from django.http import HttpResponse***

***def detail(request, question\_id):***

***return HttpResponse("You're looking at question %s." % question\_id)***

***def helloWorld(request, namae):***

***return HttpResponse("Hello world %s!" % namae)***

***def index(request):***

***#add code here***

***#https://docs.djangoproject.com/en/1.11/intro/tutorial03/#a-shortcut-render***

***return render(request, 'tbc\_sri\_app/index.html')***

1. Created tbc\_sri\_app/urls.py

to define URL and Views mapping

***from django.conf.urls import include, url***

***from django.contrib import admin***

***#***

***from . import views***

***urlpatterns = [***

***# Examples:***

***# url(r'^$', 'tbc\_sri.views.home', name='home'),***

***# url(r'^blog/', include('blog.urls')),***

***url(r'^admin/', include(admin.site.urls)),***

***url(r'^(?P<question\_id>[0-9]+)/$', views.detail, name='detail'),***

***url(r'^(?P<namae>[^0-9])/$', views.helloWorld, name='helloWorld'),***

***url(r'^$', views.index, name='index'),***

***]***

1. Updated tbc\_sri/urls.py to map with tbc\_sri\_app/urls.py mapping

***#***

***url(r'^tbc\_sri\_app/', include('tbc\_sri\_app.urls')),***

# GRID INTRO

## Grid tool to use

After comparing jsgrid (<http://js-grid.com/demos/>) and jqgrid (<http://www.guriddo.net/demo/guriddojs/>) decided to go with jsgrid. Both have somewhat recent updates and not that great HOWTOs. Just jsgrid looked more pleasant to the eyes. But it may not have CSV download so need to keep an eye on that.

## Download jsgrid package

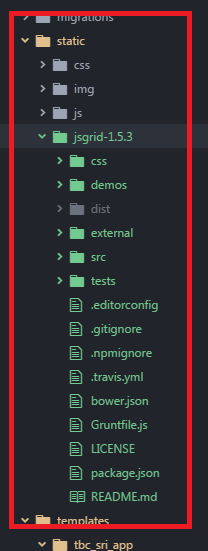
<http://js-grid.com/getting-started/>

Download the **SOURCE** and place in the static folder

<https://stackoverflow.com/questions/1532875/django-javascript-files>

Wasn’t sure of any conventions so simply placed the folder under static

tbc\_sri\_app\static\ jsgrid-1.5.3



### Ilust2. New static folder with JSGrid files

## Check settings.py

<https://scotch.io/tutorials/working-with-django-templates-static-files#toc-settings-for-managing-static-files>

***INSTALLED\_APPS = (***

***…***

***'django.contrib.staticfiles',***

…

***STATIC\_URL = '/static/'***

***STATICFILES\_DIRS = (***

***os.path.join(BASE\_DIR, 'static'),***

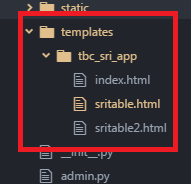
***)***

***STATIC\_ROOT = os.path.join(BASE\_DIR, 'staticfiles')***

Not sure how to check but I believe I have done all I can to include the JSGrid files. Next I will create a test page to start playing with.

## Create a test page

Create a templates/tbc\_sri\_app folder and create sritable.html in it



### Ilust3. Templates folder with index.html

I copied and pasted the jsgrid-1.5.3/demos/basic.html into the sritable.html.

Added reference to the static folder at the top

***{% load static %}***

Changed all the CSS and JS references to the static folder

***href="{% static "jsgrid-1.5.3/demos/demos.css" %}"***

## Update urls.py and views.py with the test pages

**Urls.py**

***url(r'^sritable$', views.sritable, name='sritable'),***

**Views.py**

***def sritable(request):***

***return render(request, 'tbc\_sri\_app/sritable.html')***

Now that I got the demo grid working, the next step is to connect the grid to the model

# CONNECTING THE MODEL AND THE GRID

<http://js-grid.com/docs/#grid-controller>

Trying to figure out how to create a model output object and pass that to the JSGrid.

Following the two videos, decided to create the data dictionary in views.py and pass to HTML

<https://www.youtube.com/watch?v=0HVwUQ0Ok7Y&list=PL6gx4Cwl9DGBlmzzFcLgDhKTTfNLfX1IK&index=14>

<https://www.youtube.com/watch?v=b0d09mYsORs&index=13&list=PL6gx4Cwl9DGBlmzzFcLgDhKTTfNLfX1IK>

<http://zetcode.com/articles/jsgridservlet/>

## loadData

Created yet another HTML called sritable3.html and update views.py

***def sritable3(request):***

***#load in the template and create a variable as a reference to the #template***

***template = loader.get\_template('tbc\_sri\_app/sritable3.html')***

***return render(request, 'tbc\_sri\_app/sritable3.html')***

urls.py also updated

***url(r'^sritable3$', views.sritable3, name='sritable3'),***

then created another view to query the model and return a JSON data

***def myLoadData(request):***

***#run the query***

***myQuerySet = lnos\_statusPipeLine.objects.values\_list('mbol','container')***

***#create an empty dict to put the data in***

***response\_data = {}***

***try:***

***response\_data['myKey'] = 'Success'***

***response\_data['myData'] = list(myQuerySet)***

***except:***

***response\_data['myKey'] = 'Failure'***

***response\_data['myData'] = 'Failed to query data.'***

***return HttpResponse(json.dumps(response\_data), content\_type="application/json")***

and updated the urls.py accordingly

***url(r'^myLoadData$', views.myLoadData, name='myLoadData'),***

I was able to confirm the query itself is working by directly calling this ***myLoadData*** URL.

So next I tried calling the myLoadData from the HTML.

***<script>***

***console.log("entering script");***

***$(function() {***

***//$("#jsGrid") refrences the <div> above***

***console.log("entering function")***

***$("#jsGrid").jsGrid({***

***height: "70%",***

***width: "100%",***

***filtering: true,***

***editing: true,***

***inserting: true,***

***sorting: true,***

***paging: true,***

***autoload: true,***

***pageSize: 15,***

***pageButtonCount: 5,***

***deleteConfirm: "Do you really want to delete the client?",***

***controller: {***

***loadData: function(myData){***

***console.log("entering loadData")***

***return $.ajax({***

***type: "GET",***

***url: 'myLoadData',***

***data: myData,***

***datatype: "JSON",***

***success: function(data){***

***console.log('successss', data);***

***}***

***});***

***}***

***},***

***fields: [***

***{ name: "mbol", type: "text", width: 150 },***

***{ name: "container", type: "text", width: 50 },***

***{ type: "control" }***

***]***

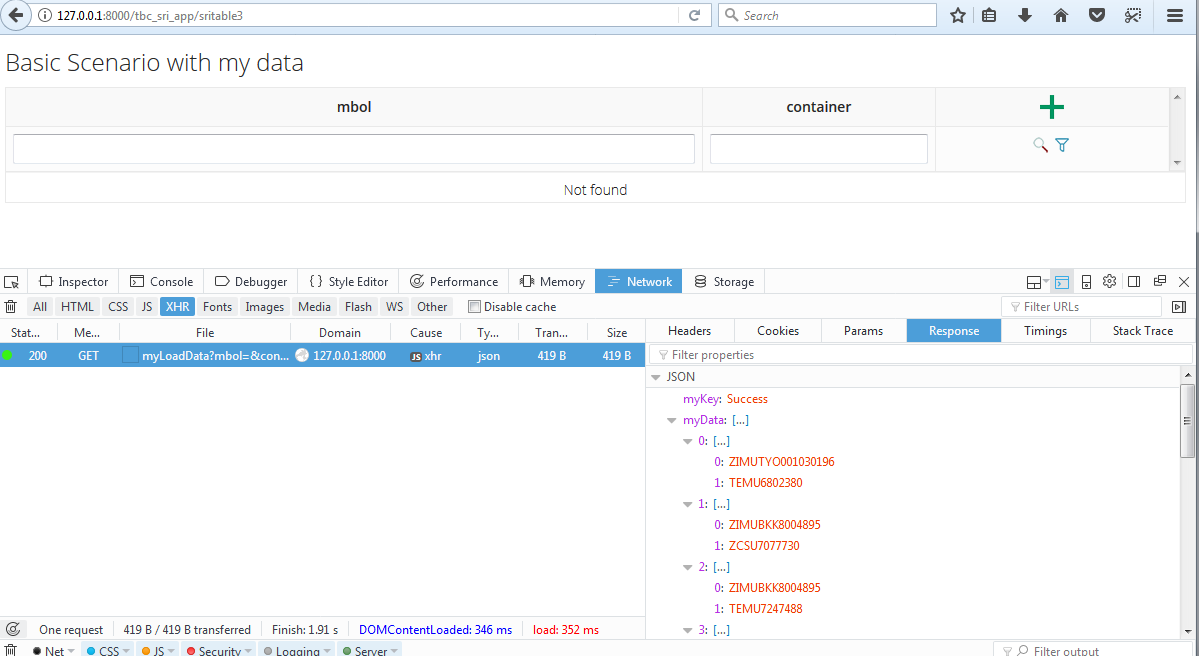
***});***

***});***

***</script>***

I now get the JSGrid table but is empty and says “Not Found”

Checked the **Network** tab and found the XHR GET. Looking into its **Response**, the data is there so it did make its way back to the HTML.



So need to figure out why the data is not making it into the grid. I have been basing on the demos/basic.html but reading this, maybe I should be referring to the OData.html in stead.

<https://github.com/tabalinas/jsgrid/issues/482>

But on the second look, the OData example table is not mutable…

Finally got it working.

Views.py

***def myLoadData(request):***

***#run the query***

***myQuerySet = lnos\_statusPipeLine.objects.values('mbol','container')***

***#cast from Queryset to “list” array***

***try:***

***response\_data = list(myQuerySet)***

***except:***

***response\_data = 'Failed to query data.'***

***#convert the list to JSON and return as HTTP response***

***return HttpResponse(json.dumps(response\_data), content\_type="application/json")***

sritable3.html

***loadData: function(){***

***var d = $.Deferred();***

***$.ajax({***

***url: 'myLoadData',***

***type: 'GET',***

***contentType : "application/json; charset=utf-8",***

***dataType: 'json',***

***}).done(function(response){***

***console.log("just response", response)***

***d.resolve(response);***

***}).fail(function(response){***

***console.log("fail", responsef)***

***d.reject();***

***});***

***return d.promise();***

***},***

## updateItem

Now that I can retrieve and display data form the backend, the next step is to be able to update them. Found this article which may shed some insight.

<http://zetcode.com/articles/jsgridservlet/>

Also this video on how to establish connection with DB

<https://www.youtube.com/watch?v=o-vsdfCBpsU>

According to this link I found, I can test the updated data I am trying to send by simply doing

<https://github.com/tabalinas/jsgrid/issues/793>

***updateItem: function(item){***

***console.log("updateItem data", item);***

***}***

I created a view for upload.

***def myUpdateData(request):***

***myDict = requests.POST***

***print(myDict)***

Which returned my “Update failed” message.

Looking on the DOS prompt, there is a message

***Not Found: /tbc\_sri\_app/myUpdateData***

***[08/Nov/2017 16:01:18] "PUT /tbc\_sri\_app/myUpdateData HTTP/1.1" 404 3511***

404 reminds me I forgot to update urls.py

***urlpatterns = [***

***…***

***url(r'^myUpdateData$', views.myUpdateData, name='myUpdateData'),***

***]***

Now I get

***Forbidden (CSRF token missing or incorrect.): /tbc\_sri\_app/myUpdateData***

***[08/Nov/2017 16:07:13] "PUT /tbc\_sri\_app/myUpdateData HTTP/1.1" 403 2502***

Adding this line to **updateItem** function in my HTML got rid of the error

***headers: {'X-CSRFToken': '{{ csrf\_token }}'},***

Now I get

***AttributeError: module 'requests' has no attribute 'POST'***

***[08/Nov/2017 17:58:17] "PUT /tbc\_sri\_app/myUpdateData HTTP/1.1" 500 18331***

Eventually realized that all thee “request” (without the ;last ‘s’) was referring to the parameter in my function in views.py:

***def myUpdateData(request):***

So combining with the info on this page, I was able to figure out how to get the Request details

<https://docs.djangoproject.com/en/dev/ref/request-response/#jsonresponse-objects>

So added the line below in red into my HTML file. Response is the entire ***HttpResponse*** so plucks out the ***responseText***

***updateItem: function(item){***

***…***

***}).fail(function(response){***

***console.log("Update failed response", response.responseText)***

***d.resolve(previousItem);***

***});***

The in views.py I tried playing with different Request handling

***def myUpdateData(request):***

***if request.method == 'PUT':***

***inMethod = "method: " + request.method + ". "***

***inMIME = "MIME: " + request.content\_type + ". "***

***return HttpResponse(inMethod + inMIME)***

***else:***

***return HttpResponse("response no update")***

Which returns…cool!

***Update failed response method: PUT. MIME: application/json.***

I tried adding in

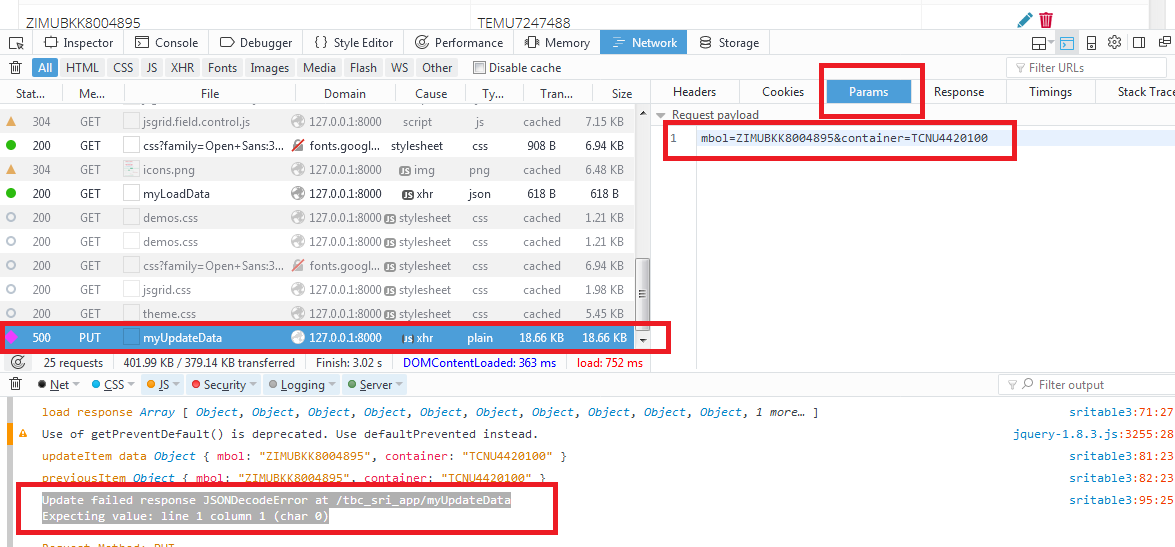
***inJSON = "JSON: " + json.loads(request.body) + ". "***

which is giving me this error

***Update failed response JSONDecodeError at /tbc\_sri\_app/myUpdateData***

***Expecting value: line 1 column 1 (char 0)***

Looking at the ***Network*** => ***Params***tab, maybe the payload is not coming through as JSON even though MIME says ‘json’?



Tried this in views.py

***return HttpResponse(request.body)***

and got

***Update failed response mbol=ZIMUBKK8004895&container=dafafadfafda***

So the payload is definitely not JSON. So ow tried changing my **uploadItem** in the HTML and got rid of these two lines:

***contentType : "application/json; charset=utf-8",***

***dataType: 'json',***

Now I get

***update response mbol=ZIMUBKK8004895&container=dsfafsdfsdf***

This did the trick to turn the payload into JSON in the **updateItem**

<https://briancaffey.github.io/2017/07/22/posting-json-data-with-ajax-to-django-rest-framework.html>

***data: JSON.stringify(item),***

But printing to the console, I noticed the ‘b’ which means this is a byte literal

***b'{"pk":24,"mbol":"ZIMUBKK8004895","container":"daf"}'***

So need to convert this to string then to JSON

<https://stackoverflow.com/questions/40059654/python-convert-a-bytes-array-into-json-format>

***strInData = request.body.decode("utf-8")***

Which produces…

***{"pk":24,"mbol":"ZIMUBKK8004895","container":"daf"}***

Took it one step further to experiment

***print("incoming data => ", request.body)***

***#b'{"pk":24,"mbol":"ZIMUBKK8004895","container":"daf"}'***

***strInData = request.body.decode("utf-8")***

***#{"pk":24,"mbol":"ZIMUBKK8004895","container":"daf"}***

***listInData = json.loads(strInData)***

***#{'pk': 24, 'mbol': 'ZIMUBKK8004895', 'container': 'das'}***

***jsonInData = json.dumps(listInData)***

***#{"pk": 24, "mbol": "ZIMUBKK8004895", "container": "das"}***

But still no idea how to get this into the DB…

# Branching to RESTful01

Reading this post led me to taking another look at RESTful at this particular stage

<https://stackoverflow.com/questions/21797436/django-how-to-update-model-field-from-json-data>

which ten led me to <http://www.django-rest-framework.org/#topics> then to <https://code.tutsplus.com/tutorials/beginners-guide-to-the-django-rest-framework--cms-19786>

***>pip install djangorestframework***

***Collecting djangorestframework***

***Downloading djangorestframework-3.7.3-py2.py3-none-any.whl (1.5MB)***

***100% |████████████████████████████████| 1.5MB 445kB/s***

***Installing collected packages: djangorestframework***

***Successfully installed djangorestframework-3.7.3***

Update **settings.py** with

***INSTALLED\_APPS = (***

***…***

***'rest\_framework',***

***)***

Create **tbc\_sri\_app\serializers.py**

***from rest\_framework import serializers***

***from .models import lnos\_statusPipeLine***

***…***

***class lnosStatusPipeLineSerializer(serializers.ModelSerializer):***

***class Meta:***

***model = lnos\_statusPipeLine***

***fields = ('pk', 'mbol', 'container')***

Update **views.py**

***from rest\_framework import generics***

***from tbc\_sri\_app.serializers import lnosStatusPipeLineSerializer***

***…***

***class myRest(generics.ListAPIView):***

***model = lnos\_statusPipeLine***

***serializer\_class = lnosStatusPipeLineSerializer***

Update **urls.py**

***from rest\_framework.urlpatterns import format\_suffix\_patterns***

***from tbc\_sri\_app import views #for the rest framework***

…

***url(r'^myRest$', views.myRest.as\_view(), name='myRest'),***

So to test it out went to the shell

***>python manage.py shell***

***>>> from tbc\_sri\_app.models import lnos\_statusPipeLine***

***>>> from tbc\_sri\_app.serializers import lnosStatusPipeLineSerializer***

***>>> foo = lnos\_statusPipeLine.objects.get(pk=25)***

***>>> serialized = lnosStatusPipeLineSerializer(foo)***

***>>> serialized.data***

***{'pk': 25, 'mbol': 'ZIMUBKK8004895', 'container': 'TEMU7247488'}***

Looking god so far…except the single quote implies this is not JSON (yet).

Tried on the browser <http://127.0.0.1:8000/tbc_sri_app/myRest>

And got

***AssertionError at /tbc\_sri\_app/myRest***

***'myRest' should either include a `queryset` attribute, or override the `get\_queryset()` method.***

Looking at the example on <http://www.django-rest-framework.org/api-guide/generic-views/> seems I need to specify the **queryset** in my views class. So updated **views.py**.

***class myRest(generics.ListAPIView):***

***model = lnos\_statusPipeLine***

***serializer\_class = lnosStatusPipeLineSerializer***

***queryset = lnos\_statusPipeLine.objects.all()***

and now I get **HTTP 200 OK** along with the data from my model.

## JSGrid with REST

Now to see how JSGrid works with REST, I went back to my HTML and simply changed the **url**

***loadData: function(){***

***var d = $.Deferred();***

***$.ajax({***

***url: 'myRest',***

…and it worked (the grid loads fine)!!! And for some reason, the myUpdateData started working…But it is not the REST solution so still need to keep working on a RESTful solution.

## PUT Request

Put request is a bit trickier because I have to specify which model object I need to update. After quite a bit of research on Views, I settled on the following.

Update **views.py** (all is identical except the View)

***class myPutRest(generics.RetrieveUpdateDestroyAPIView):***

***model = lnos\_statusPipeLine***

***serializer\_class = lnosStatusPipeLineSerializer***

***queryset = lnos\_statusPipeLine.objects.all()***

Update **urls.py** with the new view and passing the pk

***url(r'^myPutRest/(?P<pk>[0-9]+)$', views.myPutRest.as\_view(), name='myPutRest'),***

Then finally update the **updateItem** controller in the **HTML** to pass the pk

***updateItem: function(item){***

***…***

***var thisPK = item["pk"]***

***console.log("thisPK is now ", thisPK)***

***$.ajax({***

***url: 'myPutRest/'+thisPK,***

***type: 'PUT',***

***…***

I see the updateItem shows the new value

***updateItem data Object { pk: 25, mbol: "mbol01", container: "container02" }***

and the right pk is being passed but somehow the update response shows the original values

***update response Object { pk: 25, mbol: "mbol01", container: "container01" }***

Would like to see what is being passed to the myPutRest…

Added this code to **views.py** see what data is being passed and returned

***class myPutRest(generics.UpdateAPIView):***

***…***

***def update(self, request, \*args, \*\*kwargs):***

***inObject = self.get\_object()***

***inSerialized = self.get\_serializer(inObject, data=request.data)***

***inSerialized.is\_valid(raise\_exception = True)***

***print("RAW ", inSerialized)***

***print("DATA ", inSerialized.data)***

***return Response(inSerialized.data)***

which gives me…

***RAW lnosStatusPipeLineSerializer(<lnos\_statusPipeLine: lnos\_statusPipeLine object>, context={'request': <rest\_framework.request.Request object>, 'format': None***

***, 'view': <tbc\_sri\_app.views.myPutRest object>}, data=<QueryDict: {'{"pk":25,"mbol":"mbol01","container":"container02"}': ['']}>):***

***pk = IntegerField(label='ID', read\_only=True)***

***mbol = CharField(allow\_blank=True, max\_length=24, required=False)***

***container = CharField(allow\_blank=True, max\_length=24, required=False)***

***DATA {'pk': 25, 'mbol': 'mbol01', 'container': 'container01'}***

Which leads me to believe the serializer maybe doing something wrong. I am suspecting that instead of looking up the pk and updating it, it is looking up the pl and returning it.

# Branching to RESTful02

PUT was simply not working and all of this condensed code that does everything for you behind the scenes was not helping me understand how things are working, decided to roll back to before I started implementing REST so I can start REST from scratch.

## Serializer

<http://www.django-rest-framework.org/tutorial/1-serialization/>

Added “rest\_framework’ in **settings.py**.

1. **INIT:** Create virtenv, project, app and model as usual
2. **SERIALIZE:** Create **ModelSerializer**.

Create class specifying the

* Model
* Fields

This serializer will handle

* 1. Determine set of model fields to instanciate for serializing
  2. Simple default implementation for create() and update() methods

1. **VIEW**
   1. HTML
   2. GET or POST or PUT

Pass the request

<WSGIRequest: PUT '/tbc\_sri\_app/myUpdateData02/25'>

If GET

Obtain QuerySet from the model

Serialize

Return in JSON

If POST/PUT

Parse the stream into Python native data type

Then turn that into a dictionary of validated data

{'pk': 25, 'mbol': 'mbol01', 'container': 'container02'}

Serialize (pass the original and new objects for PUT)

lnosStatusPipeLineSerializer(<lnos\_statusPipeLine: lnos\_statusPipeLine object>, data={'pk': 25, 'mbol': 'mbol01', 'container': 'container02'}):

1. **URL**

Update **urls.py**

1. **JS**

Make sure to pass the pk for PUT

***updateItem: function(item){***

***var d = $.Deferred();***

***var thisPK = item["pk"]***

***$.ajax({***

***url: 'myUpdateData02/' + thisPK,***

## Requests and Response

<http://www.django-rest-framework.org/tutorial/2-requests-and-responses/>

from django.http import HttpResponse #replaced by .Response

from django.http import JsonResponse #replaced by .Response

replaced by…

from rest\_framework.response import Response

**Error:**

***AssertionError: .accepted\_renderer not set on Response***

Don’t froget…

1. from rest\_framework.decorators import api\_view
2. @api\_view(['GET', 'POST', ‘PUT’, ‘DELETE’])

**@api\_view (views.py)**

<http://www.django-rest-framework.org/tutorial/2-requests-and-responses/#pulling-it-all-together>

#No longer necessary to manually convert input stream into dict after using the api\_view

data = JSONParser().parse(request) #dict at this point

serializer = lnosStatusPipeLineSerializer(data = data)

#instead simply call the serializer passing the request.data as below

serializer = lnosStatusPipeLineSerializer(data=request.data)

or

serializer = lnosStatusPipeLineSerializer(cargo, data = request.data)

After this conversion, PUT stopped working again and would not update.

Compared the outputs of the two…

If I am manually parsing the request into dict I get a proper dict

data = JSONParser().parse(request) #dict at this point

{'pk': 25, 'mbol': 'mbol01', 'container': 'container02'}

If I pass request.data instead then I get QueryDict

serializer = lnosStatusPipeLineSerializer(cargo, data = request.data)

<QueryDict: {'{"pk":25,"mbol":"mbol01","container":"container02"}': ['']}>

I suppose this is expected because it is basically saying that

JSONParse().parse(request) <> request.data

…even though the tutorial says

*…request.data can handle incoming json requests,…*

But apparently it is not.

I tried casting the QueryDict to dict

serializer = lnosStatusPipeLineSerializer(cargo, data = dict(request.data))

and now I get

{'{"pk":25,"mbol":"mbol01","container":"container02"}': ['']}

# References

## Django 1.11 official Tutorial

<https://docs.djangoproject.com/en/1.11/intro/tutorial01/>

## Django Import\_Export

<http://django-import-export.readthedocs.io/en/latest/api_resources.html>)

## How to serve static files including collect statics

<https://scotch.io/tutorials/working-with-django-templates-static-files#toc-settings-for-managing-static-files>

<https://stackoverflow.com/questions/12031825/how-to-set-up-django-website-with-jquery>

## JSGrid

<https://github.com/tabalinas/jsgrid#requirement>

<https://github.com/tabalinas/jsgrid-django>

<http://js-grid.com/demos/>

<http://js-grid.com/getting-started/>

http://zetcode.com/articles/jsgridservlet/

## JQGrid

<http://www.guriddo.net/demo/guriddojs/>

## Views

<http://www.django-rest-framework.org/api-guide/generic-views/>

<http://www.django-rest-framework.org/tutorial/3-class-based-views/>