PRASHANT L

PRODUTION SUPPORT TOOL TECHNICAL DESIGN DOCUMENT

**Release History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Release Date** | **Components** | **Description** | **Section #** |
| May, 2021 | Prod-support-tool technical document | Prod Support Tool technical design document | 1 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Technical Details

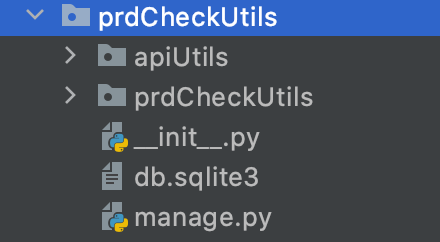
The tool is created using python based framework Django , to start with create a separate python virtual environment and pip install django

Django Documentation and Tutorial Link : [Here](https://docs.djangoproject.com/en/3.2/intro/tutorial01/)

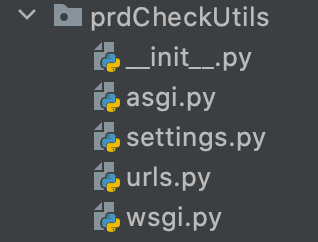
Django Project Structure has 2 components

1. **Project**

Project is main folder structure which has files needed for entire application Create project **django-admin startproject <PROJECT\_NAME>,** here project name is **prdCheckUtils** this creates basic project structure, this creates main file **manage.py** which is needed to run Django project as UI app and sub folder with same name as project



Subfolder contains important files like **settings.py**, **urls.py**



**settings.py** contains all the project settings

**urls.py** contains all the redirects/urls in app

1. **App**

One Project can have multiple Apps, application modules can be further broken down to apps, to create app: **python manage.py startapp <APP NAME>** here app name is **apiUtils**

App folders has important files like **urls.py, views.py, model.py, forms.py and folders like static, templates**

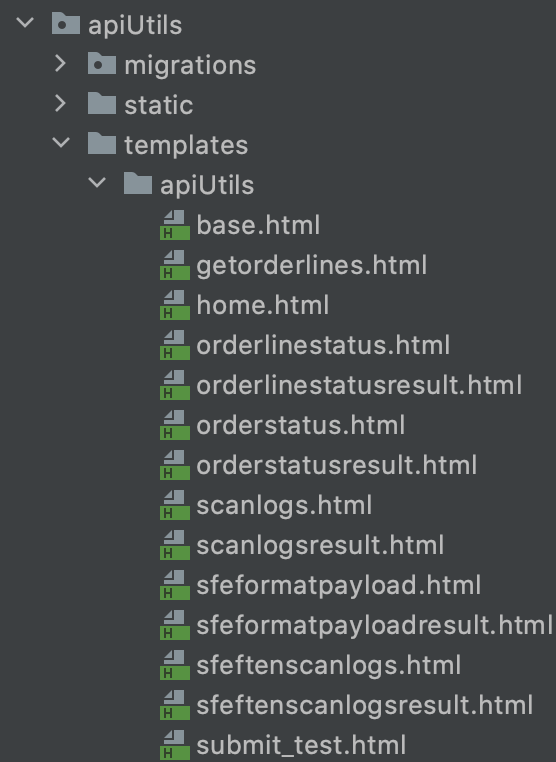
urls.py 🡪 This is will have all redirects/routes

views.py 🡪 every route defined in urls.py is associated with a function, which are defined here, each route has a function/action defined against it

forms.py 🡪 this file has all UI fields defined, each UI page is a function and each UI fields are variables

static 🡪 this folder has all static files like .js, .css and image files

template 🡪 this folder has sub folder with <APP\_NAME> which has all the html files defined



HTML files follows [Jinja](https://jinja.palletsprojects.com/) template, where values returned from views functions are captured in HTML and rendered on to page

Either can have single page/route to handle input and output or there can be separate routes/page one for input and one for output , separate output page is preferred where there multiple results or pagination is needed

**Few Important HTML files**

**base.html** 🡪 This is a base html template , which has common objects that appears on all html pages , other html pages would inherent this base html , by including base.html in every html file , like left panel of page/menu which should appear on all pages, so this is panel is defined in base.html

**home.html** 🡪 This is used to render home page , inherited from base.html

Example Creating an Utility

**Utility Name :** Get Order Status

**HTML Pages** : orderstatus.html (Input Page)

orderstatusresult.html (Result Page)

**URL’s(urls.py):** orderstatus/ (Input Page)

orderstatusresult/ (Result Page)

**VIEWS(views.py):** orderStatus (Input Page)

orderStatusResult (Result Page)

**FORMS(forms.py):** OrderStatusForm (Input Page)

**Input Page :**

Graphical user interface, text, application

Description automatically generated

1. **FORMS**

Files associated , forms.py this file is used to create page layout/ page view,

entire page is associated with a class and page fields are variables inside a class , here function associated is **OrderStatusForm.**

**Text

Description automatically generated**

**For Owner and Env Drop Down , create a list if tuples**

ENV\_CHOICE = [

('dev', 'dev'),

('test', 'test'),

('stage', 'stage'),

('pl465j', 'pl465j'),

('prd', 'prd'),

]

OWNER\_CHOICE = [

('WarnerMedia', 'WarnerMedia'),

('DAS', 'DAS'),

('Adcuratio', 'Adcuratio'),

('OperativePil', 'OperativePil'),

]

Text

Description automatically generated

**Filed Definitions**

**Text

Description automatically generated**

1. **env:** Env Filed is a drop down, **CharFiled**, defined as **Select** widget which takes in list as choice, ENV\_CHOICE is provided as input, label given to filed is **ENV** which is displayed in page
2. **owner:** Owner Filed is a drop down, **CharFiled**, defined as **Select** widget which takes in list as choice, OWNER\_CHOICE is provided as input, label given to filed is **OWNER** which is displayed in page
3. **orderIds:** OrderId’s filed is a text filed**, CharFiled,** defined as **TextInput** widget, required is set to **True** as it is a mandatory field, label given to filed is **ORDERID(s)** which is displayed in page
4. **VIEWS**

views.py is file associated with views, this is responsible to render forms as html. object and also receive input from url’s/frontend and translate inputs and process the requests and redirects to result url or returns values to html, which is later rendered as jinja templates, from object that is returned , usually json is returned from views

views are associated with function, here we have two functions

1. **orderStatus:** This function is associated with input page, view class is imported as form and form fields is internally converted to html and retuned as response which is then rendered to orderstatus.html with fields defined in forms , which is input page, associated cookies are deleted
2. **orderStatusResult:** This function is used to receive input, when input form is submitted, input fields are validated, if submitted form is invalid, page is redirected to input page itself with error message displayed , if submitted form is valid , input field values received are set as cookies , which is used for pagination

based on input received, method to process the request is called and based on response received , if response was error then function redirects to input page with error , if successful function redirects to result url and renders the json received , and these results are paginated, through django has predefined pagination functions

1. **URLS**

urls.py is file associated with url’s , all url’s are defined in a predefined list called urlpatterns , list item is a path object , which takes 3 variables i.e.

1. url/route
2. view function associated with it
3. name or url to be referenced in view function and html (using jinga template {% url <URL\_NAME> %})

Example

urlpatterns = [

path('home/', views.HomePage, name='home\_page'),

path('orderstatus/', views.orderStatus, name='orderstatus'),

path('orderstatusresult/', views.orderStatusResult, name='orderstatusresult'),

path('orderlinestatus/', views.orderLineStatus, name='orderlinestatus')

]

Here we have 2 url’s defined.

1. **orderstatus:** This url/route is associated with input page, and view function associated this url is orderStatus, which is responsible for displaying input page. Example **path ('orderstatus/', views.orderStatus, name='orderstatus')**
2. **orderstatusresult:** This url/route is associated with result/response page, and view function associated this url is orderStatusResult, which is responsible for rendering json response received/returned from view function  **path ('orderstatusresult/', views.orderStatusResult, name='orderstatusresult')**

**Other Custom Files**

**constants.py** 🡪 This file has all constants like Rest API backend url’s , user and passwords , this file is stored as a secretes in aws/eks env

**templatetags/extras.py 🡪** This file contains user defined functions that is used in HTML pages , here function (pretty\_json) to format/pretty print json received from Invidi and render to pages for viewing

**Creating a new Utility**

Let’s say we need to create a new utility for icd-63/scte data

1. Create 2 html pages one for input page and another for result i.e. scteinput.html, scteresult.html
2. create a new class in forms.py with these fields Owner,Env drop down and following numeric input fields
3. WindowID
4. AvailID
5. AllocationID
6. SlotID
7. create 2 view functions scteInput and scteResult one to handle/render input page and other page to display/render json results
8. create 2 urls/redirect one to handle input page and one to handle results
9. here results are not paginated for single Id gives json with 2 keys dbdata and icddata, which results are json, so per id need to render 2 json
10. modify constants.py to add new icd-63 related url
11. add scte utility to menu panel, as a new menu item, modify base.html

Note:This can be done with a single html page with both input and result in single page, like in getOrderLine , since response/result will not have too many json item to render and would not need pagination

**Files To be Modified**

1. Add new HTML files in **app\_prod-support-ui-tool/app/prdCheckUtils/apiUtils/templates/apiUtils** directory
2. Modify **app\_prod-support-ui-tool/app/prdCheckUtils/apiUtils/urls.py** file add new routes to urlpatterns list
3. Modify **app\_prod-support-ui-tool/app/prdCheckUtils/apiUtils/templates/apiUtils/base.html,** add new route created to menu
4. Modify **app\_prod-support-ui-tool/app/prdCheckUtils/apiUtils/forms.py** create new class as per page layout(fields)
5. Modify **app\_prod-support-ui-tool/app/prdCheckUtils/apiUtils/views.py** add new functions to render view class as input page and to receive input from page process and render results

\*\*If a new user needs access to tool via Global Logon, attuid needs to be added in a particular permission file, Need to contact Jim to get it added