Starting a project basics

1. Making virtual environment for projects.

* Download with command: pip install virtualenv
* Setup in folder: virtualenv {name\_of\_env}
* Activate env: todoenv\scripts\activate
* Now its activated so we can install other things in it

1. Installing Django framework.

* Install Django: pip install Django
* Setup Django app: django-admin startproject {name\_of\_project}
* Now put the virtual environment into Django app folder
* Now to run server: python manage.py runserver

1. Now to create an app inside Django project:

* Command: python manage.py startapp {name\_of\_app}

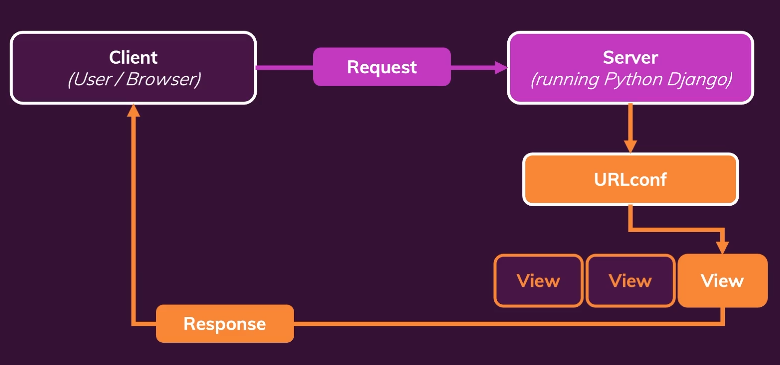
1. Add ApiConfig method from api apps.py into the main settings.py inside installed\_Apps list.

‘api.apps.ApiConfig’

1. Now we setup the view of the api app inside views.py file.

* For this import: from django.http import JsonResponse
* And then create view:
* def getRoutes(*request*):
* *return* JsonResponse('Our API', *safe*=False)

1. Now URLs/Routes which ensure that certain results are achieved when certain URLs are entered by the user and Views are logic that is executed for different URLs (and HTTP methods), basically code that handles requests and return responses.
2. Views are responsible for processing requests and creating a response.
3. Setup urls file in api app for routes and add it to main urls.py file in project folder.
   * from django.urls import path
   * from . import views
   * urlpatterns = [
   * path('', views.getRoutes, name="routes"),
   * ]
   * Now inside main urls.py file:
   * from django.contrib import admin
   * from django.urls import path, include
   * urlpatterns = [
   * path('admin/', admin.site.urls),
   * path('/', include('api.urls')),
   * ]



1. To create dynamic URL path we can use dynamic segment as: here ‘<month>’ is

* urlpatterns = [
* # path('january', views.jan),
* # path('feb', views.feb),
* path('<month>', views.monthly\_challenge)
* ]
* While in views it will work like this:
* def monthly\_challenge(*request*, *month*):
* text = None
* *if* *month* == 'jan':
* text = 'jan bolte'
* *elif* *month* == 'feb':
* text = 'feb bolte'
* *elif* *month* == 'mar':
* text = 'mar bolte'
* *else*:
* *return* HttpResponseNotFound('this month is not found!')
* *return* HttpResponse(text)

1. And in Url patterns url path order matters to load whatever we wanted while working with different value types of dynamic segment.
2. Instead of working with hard coded redirect url we can use ***‘reverse’*** method using url name for redirect.
3. ***Now for creating template to render data in html files. We need create some subfolder named “Templates” in app folder inside which create another subfolder named “{same-as-app-name}”. And then we can create html files in that folder.***
4. To render app related template we can add: from django.template.loader import render\_to\_string

* response\_data = render\_to\_string("***challenges/challenge.html***")
* return HttpResponse(response\_data)

1. To use global template we need register the path in settings.py in ‘DIRS’: [ {add path here} ].
2. To make template dynamic we use DTL, as

* return render(request, "challenges/challenge.html", {
* 'text': text
* })
* We are sending arguments in curly braces named ‘context’ which can be accessed in template file.

1. Django project will automatically know where to find templates as we registered challenges app in installed app

* INSTALLED\_APPS = [
* 'django.contrib.admin',
* 'django.contrib.auth',
* 'django.contrib.contenttypes',
* 'django.contrib.sessions',
* 'django.contrib.messages',
* 'django.contrib.staticfiles',
* ***'challenges.apps.ChallengesConfig'***,
* ]

1. While working with templates, as ***‘reverse’*** method wont work in html file we can use some tags. Which is ***{% url ‘path name’ arg1 arg2 %}*** inside href.
2. To render the content conditionally we can use

* ***{% if text is not None %}***
* ***<h3>{{ text }}</h3>***
* ***{% else %}***
* ***<p>There is no challenge for this month</p>***
* ***{% endif %}***

1. We can’t use render function for 404 error as it only returns success response.
2. While working with databases in models.py we need to use commands for create table and migrating it for use.

Python manage.py migrate (for initialising)

Python manage.py makemigrations (for create modelled database)

Python manage.py migrate (for using the created model)

1. To access the admin panel. Create super user: python manage.py createsuperuser
2. Then to access database in admin panel we need to register it in admin.py file in api app folder.
   * from django.contrib import admin
   * # Register your models here.
   * from .models import Note
   * admin.site.register(Note)
   * then it will show these models in admin panel and we can add those objects there

Working with Django Rest Framework that allows us to make Django APIs.

* Using Command: python -m pip install djangorestframework
* And put it in installed apps in settings.py
  + ‘rest\_framework’

We need to add serializers to get data from database model into views. It will help us to show json data onto the view as it was object initial which we serialized.

*from* rest\_framework.serializers *import* ModelSerializer

*from* .models *import* Note

*class* NoteSerializer(ModelSerializer):

*class* Meta:

        model = Note

        fields = '\_\_all\_\_'

Now import the serializers.py file in views.py file and use.

*from* .serializers *import* NoteSerializer

@api\_view(['GET'])

def getNotes(*request*):

    notes = Note.objects.all()

    serializer = NoteSerializer(notes, *many*=True)

*return* Response(serializer.data)

After Setting up React App

We can get CORS Error as we are running on two different ports so react server wont be able to fetch data from Django port.

To fix it we need:

* Install package: Django cors header
* Inside Django terminal: pip install django-cors-headers
* Then add it to installed apps in settings.py: ‘corsheaders’
* add a middleware class:
* MIDDLEWARE = [
* ...,
* "corsheaders.middleware.CorsMiddleware",
* "django.middleware.common.CommonMiddleware",
* ...,
* ]

For adding routing to React Applications:

* We need React router dom.
* Install: npm install react-router-dom

To remove http address from the fetch() func we can put the address in package.json file as a proxy url under name tag.

"proxy": "http://127.0.0.1:8000",

While working with state updation, history doesn’t work instead we need to use useNavigate()

let navigate = useNavigate()

let handleSubmit = () => {

        console.log('NOTE:', note)

        updateNote()

        navigate('/')

    }

To get the last updated thing on top we can use

notes = Note.objects.all().order\_by('-updated')

as it will put the updated one on top of the order of the list.

Ways of connecting both apps:

* Either deploy them separately or let them connect through api calls.
* Else we can put react app inside Django backend folder.
* Run command: cd todofrontend
* Use command to build react app: npm run build
* Add code to settings.py so that Django knows the templates path.
* 'DIRS': [
* BASE\_DIR / 'todofrontend/build'
* ],
* Now we need to link static files of react to Django:
* STATICFILES\_DIRS = [
* BASE\_DIR / 'todofrontend/build/static'
* ]
* At the end import TemplateView to use those as react app on Django port.
* *from* django.views.generic *import* TemplateView
* urlpatterns = [
* path('admin/', admin.site.urls),
* path('api/', include('api.urls')),
* path('', TemplateView.as\_view(*template\_name*='index.html'))
* ]

Hash Router is used instead of BrowserRouter so that it does not mess up with urls react and Django and it will let react controls the url on Django port. It’s a quick fix but not the pretty one.

HashRouter as Router,

For adding CSS files:

* Add static folder inside app folder and inside that folder add another folder named same as app\_name.
* Then add CSS file in that folder for styling.
* Make sure we have Django.contrib.staticfiles in installed file section in settings.py file.
* Make sure we have STATIC\_URL = ‘/static/’ in settings.py.
* Then we can add:
* {% load static %}
* {% block css\_files %}
* <link *rel*="stylesheet" *href*="{% static 'challenges/challenges.css'%}">
* {% endblock %}
* To the html files.
* And then can create a global static folder where we can put base.css.
* But we need to provide the path to access that file.
* We should always use
* {% static "challenges/challenges.css" %}?{% now "U" %}
* So that it will help load css files on borwser or else ctrl+f5.