**Frontend Setting**

npm init -y (from frontend foler)

npm i webpack webpack-cli --save-dev

npm i @babel/core babel-loader @babel/preset-env @babel/preset-react --save -dev

Getting Set Up

npm i react react-dom --save-dev

npm install @material-ui/core

npm install @babel/plugin-proposal-class-properties

npm install react-router-dom

npm install @material-ui/icons

Create new folders under frontend

src

components

index.js

static

css

images

Frontend

templates

babel.config.json

webpack.config.js

Paste following in babel.config.json:

{

*"presets"*: [

[

"@babel/preset-env",

{

*"targets"*: {

*"node"*: "10"

}

}

],

"@babel/preset-react"

],

*"plugins"*: ["@babel/plugin-proposal-class-properties"]

}

Paste the following in webpack.config.js

*const* path = require("path");

*const* webpack = require("webpack");

*module*.*exports* = {

entry: "./src/index.js",

output: {

path: path.resolve(\_\_dirname, "./static/frontend"),

filename: "[name].js",

},

module: {

rules: [

{

test: /\.js$/,

exclude: /node\_modules/,

use: {

loader: "babel-loader",

},

},

],

},

optimization: {

minimize: true,

},

plugins: [

new webpack.DefinePlugin({

"process.env": {

// This has effect on the react lib size

NODE\_ENV: JSON.stringify("production"),

},

}),

],

};

In package.json, under ‘scripts’ add the following:

*"scripts"*: {

*"dev"*: "webpack --mode development --watch",

*"buid"*: "webpack --mode production"

},

Create virtual environment > conda create --name ***myenv***

Activate virtual environment > conda activate ***myenv***

Run project > python manage.py runserver

Create app > python manage.py startapp ***appname***

Create a urls.py file within the ***appname*** folder. All urls specific to the app will reside here. We will use the main project’s urls.py file only to direct URLs to the app using the url mapping.

In the settings.py under INSTALLED\_APPS, add ***appname***.apps.***appname***Config to the list

**General Instructions**

In the views.py file, each view(function) must return some HTTP response:

Example > def ***function\_name***(request):

return render (request, path, context)

(By convention the parameter is ‘request’. It can be anything)

In the main project urls.py file:

from django.conf.urls import url, path (these are missing in the starter file)

url(path, include(‘***appname***.urls’***)***

In the app folder urls.py file:

from . import views

Urlpatterns = [

path(‘browser\_url\_path’, views.***function\_name***, name=’***chosen\_name***’)

**Templates**

Create a ‘templates’ directory in the ***appname*** folder. Then create another folder in the templates folder with ***appname***

*Add the templates directory in settings.py >*

*import os*

*TEMPLATE\_DIR = os.path.join(BASE\_DIR, ‘templates’)*

*Add TEMPLATE\_DIR to TEMPLATES dictionary in settings.py*

Add code to change the title instead of hard coding it. The title is inserted in the views.py file in the respective functions as a context

Example: {% if title %}

<title>Django Blog - {{ title }}</title>

{% else %}

<title>Django Blog</title>

{% endif %}

**Base.html**

Create a base.html template in the templates/***appname***

In base.html in the <body> add the following:

{% block content %} {% endblock %}

This is the master template and avoids repetition

Remove html boilerplates from all other .html files

Insert the following in each /html file:

{% extends “***appname***/base.html” %}

{% block content %}

{% endblock content %} --> this goes at the end of the html

{%%} is used for adding python code in HTML file. Use endfor / endif etc to close the code

{{ }} is used for grabbing python objects

Add bootstrap styling if needed

Add a new folder called ‘static’ in ***appname*** directory.

Inside ‘static’ folder create another folder ***appname***

All css and javascript files will live here

At the top of base.html, insert {% load static %}

This makes the static folder available to base.html

Link a new stylesheet in the header with href={% static ‘***appname***/main.css %}

**Avoiding Hard Coding of URLs (example)**

In base.html go to the href for Home

Add “{% url ‘***chosen\_name***’ %}” (this is the name we gave to our url pattern)

**Admin**

From project folder:

python manage.py makemigrations

python manage.py migrate

python manage.py createsuperuser

Username: choose (malyaj)

User email: choose (malyaj@gmail.com)

Password: choose (mkdjango123)

**Database**

Open models.py

Every table is a Class

Every column is an attribute of this class

Example:

*class* Post(models.Model):

title = models.CharField(*max\_length*=100)

content = models.TextField()

date\_posted = models.DateTimeField(*default*=timezone.now)

author = models.ForeignKey(User, *on\_delete*=models.CASCADE)

python manage.py makemigrations

python manage.py sqlmigrate appname 0001 (numeric of the database file)

**Query Database from Django**

Python manage.py shell

From blogsite.models import Post

From django.contrib.auth.models import User

User.objects.all()

Post.objects.all()

**Create new post**

Define user

user = User.objects.filter(username=’malyaj’).first()

post\_1 = Post(title=’Blog1’, content=’First blog content’, author=user)

Post\_1.save()

**Query All Posts by a user**

Define user

user = User.objects.filter(username=’malyaj’).first()

See all posts by user

user.post\_set.all()

Create a post !!

user.post\_set.create(title=” “, content=” “)

Now we will remove the manual posts created in the views.py file and create posts by querying the Post class

*def* home(*request*):

context = {

'posts': Post.objects.all(),

}

return render(request, 'blogsite/home.html', context)

Add Model to Django Admin

Open admin.py

from .models import ***modelname (in this example it is ‘Post’)***

Refresh Django Admin

**Create User App**

This will be a new app. So create it as we create a new app

Add this app to settings.py

Open views.py in the new app and add the views

from django.contrib.auth.forms import UserCreationForm

*def* register(*request*):

form = UserCreationForm()

return render(request, 'users/register.html', {'form': form})

Create templates folder in users

Create users folder in templates

Create register.html

Inherit template and create Sign up field

{% extends "blogsite/base.html" %}

{% block content %}

<div *class*="content-selection">

<form *method*="POST">

{% csrf\_token %}

<fieldset *class*="form-group">

<legend *class*="border-bottom mb-4">Join Today</legend>

{{ form.as\_p }}

</fieldset>

<div *class*="form-group">

<button *class*="btn btn-outline-info" *type*="submit">Sign Up</button>

</div>

</form>

</div>

{% endblock content %}

Csrf token projects against attacks on user registration

Fieldset groups related elements in HTML

Form.as\_p renders the form as a paragraph

Add to this page the ability to Sign in if one is already registered

Add below to the register.html div

<div *class*="border-top pt-3">

<small *class*="text-muted">

Already a member? Sign in <a *class*="ml-2" *href*="#">Sign In</a>

</small>

</div>

Now in the main projects folder, in the urls.py file, import user.views (note that we could have also added this to our users.urls.py file

from users import views as user\_views

Add this to urlpatterns array

path('register/', user\_views.register, *name*='register')

However this form does not do anything right now because we are not doing anything with its POST request. To add a behaviour to POST do the following: