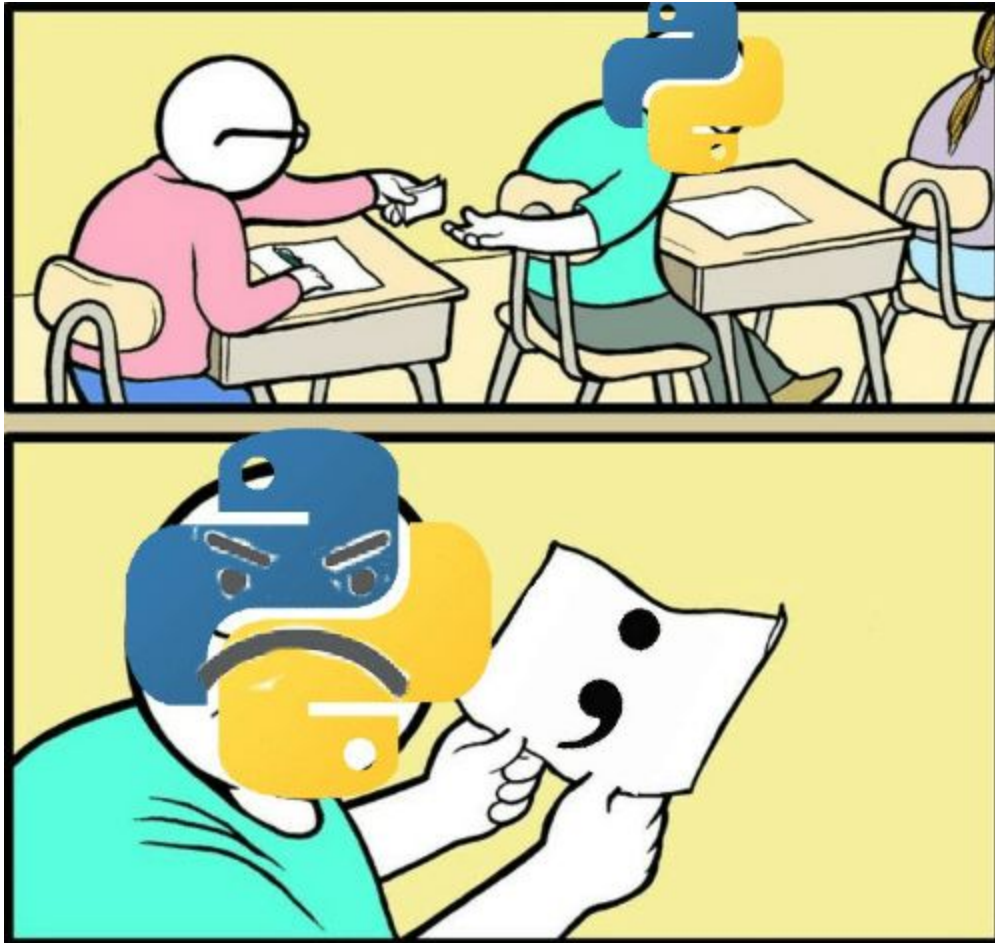


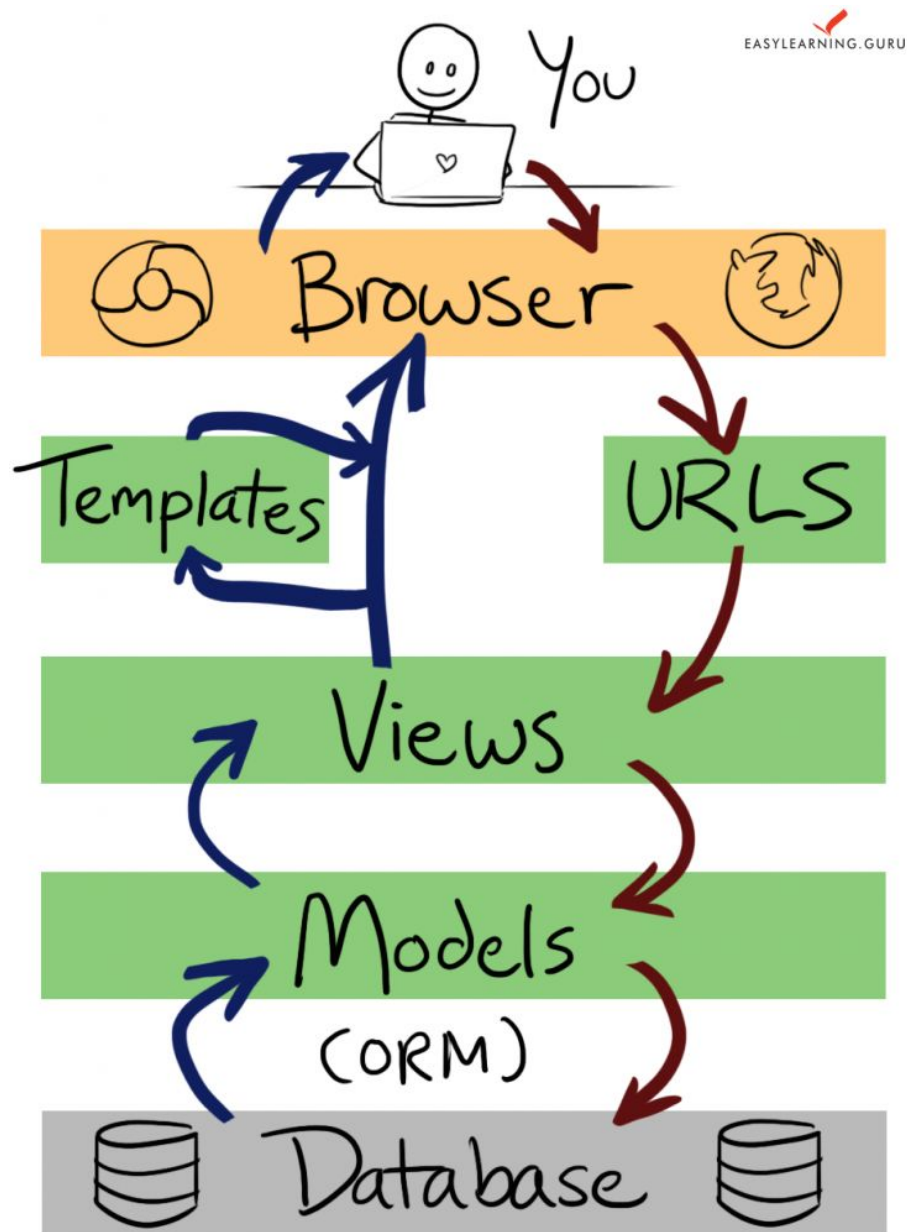
Django Project Guide

A cheat sheet provided by your course teacher :D



Python IDEs

Make the path from Django to Browser



Create an ER diagram

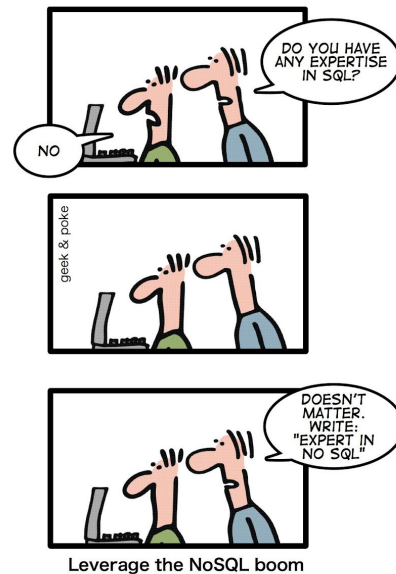
1. Find the entities or classes of the project
2. Analyse the relationship between the entities
3. Use a professional tool to create the ERD
4. Add necessary attributes to every entity

Note: ER diagram might change during the development process

Create Schema diagram

1. Use the standard rules to derive the schema diagram from ERD

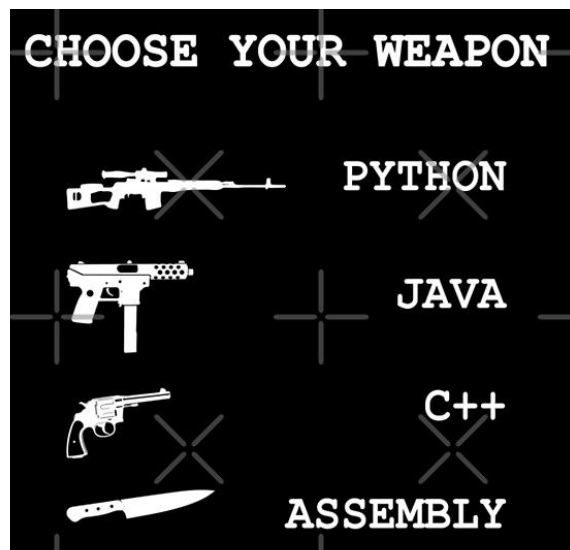
HOW TO WRITE A CV



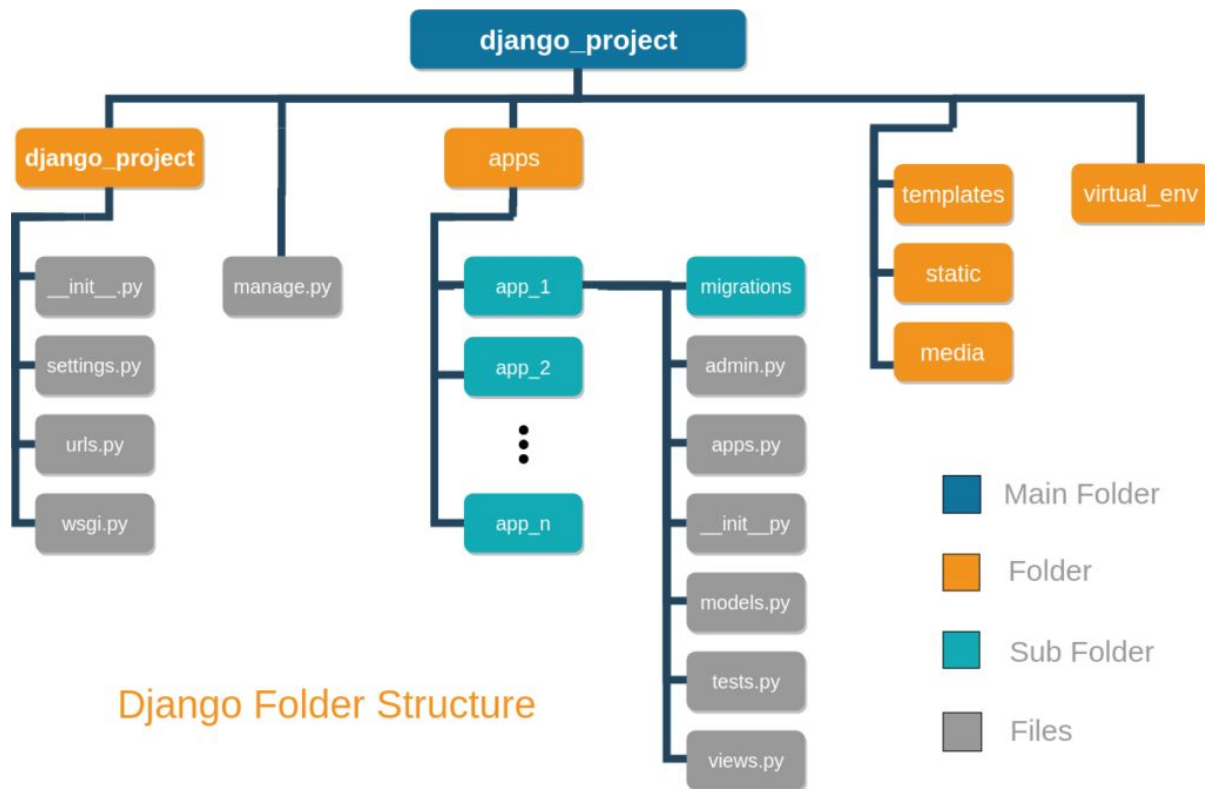
Create Project and Upload to GitHub

1. One member should create a new project (without any virtual environment)
2. Share the project with your team member and me.

[Create and Upload a Django Project to GitHub](#)



Create APPs



Divide the project into different apps based on the functionality. There is no right or wrong design here. But there are good and bad designs. Good design will help you to manage the project efficiently.

Me: *trying to save my Python code*

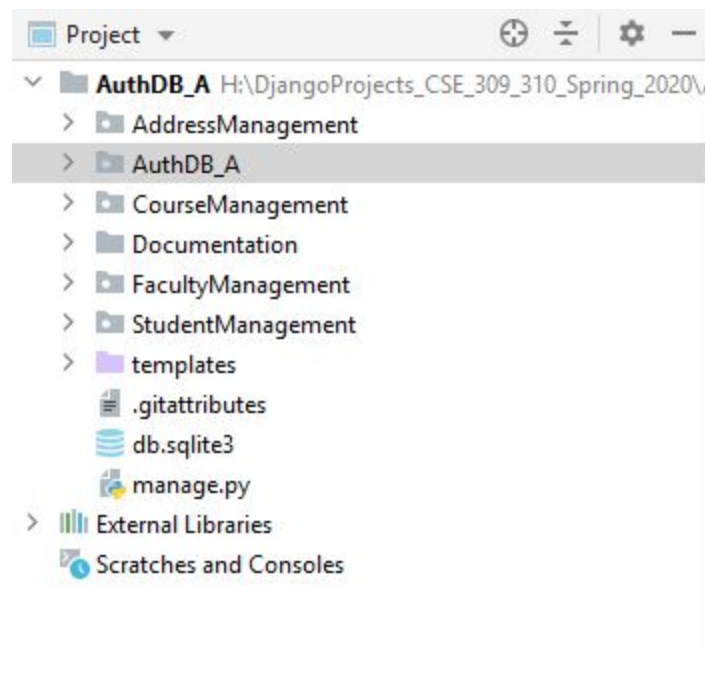
Sublime Text:



For example: If you want to develop an university management system, you might create the following apps

- StudentManagement
- FacultyManagment
- CourseManagement
- StaffManagement
- Administration
- ClubManagement
- Accounts
- EventManagerment
- Department

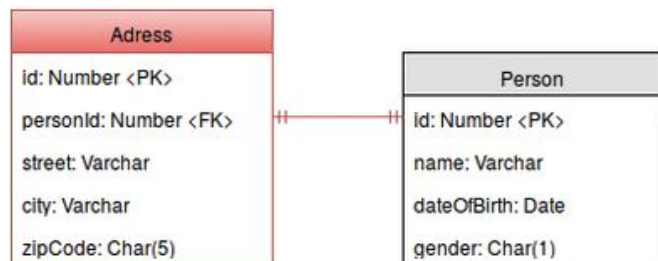
and so on



Implement models

1. Every app has its own models.py
2. Implement models according to the schema diagram (within the appropriate app)
3. Import tables from other models if necessary
4. Use foreign key attribute to create the relationships among the tables
5. Add "on delete" attribute to the foreign key function
6. Determine the dependencies

For example: consider the following tables



Person is independent as it is not pointing to another table. On the other hand, Address table is pointing to the Person table (with a foreign key personid)



StudentManagement/models.py

```
1 from django.db import models
2 from FacultyManagement.models import Advisor
3 from CourseManagement.models import Course
4 # Create your models here.
5
6 class Student(models.Model):
7     name = models.CharField(max_length=100)
8     email = models.EmailField(max_length=200, unique=True)
9     contact_no = models.IntegerField(blank=True, null=True)
10
11     advisor = models.ForeignKey(Advisor, on_delete=models.SET_DEFAULT, default=1)
12
13     # s_id = models.IntegerField(unique=True)
14     # id = models.IntegerField(primary_key=True)
15
16     def __str__(self):
17         return self.name
18
19
20 class Student_Course(models.Model):
21     student = models.ForeignKey(Student, on_delete=models.CASCADE)
22     course = models.ForeignKey(Course, on_delete=models.SET_NULL, null=True, blank=True)
23
24     def __str__(self):
25         return self.student.name + " : " + self.course.code
26
```

CourseManagement/models.py

```
1 from django.db import models
2
3 # Create your models here.
4
5 class Course(models.Model):
6     name = models.CharField(max_length=200)
7     code = models.CharField(max_length=20, unique=True)
8
9     def __str__(self):
10         return self.code
11
```


Create forms

1. Initially forms.py is not created inside an app. So, inside the app directory create a new form.py
2. Import model Class from models.py
3. Create a new class for form and inherit with ModelForm
4. Create a class Meta
5. Inside the Meta, add model and fields
6. fields == '__all__' means all fields of the model

Example:

```
from django.forms import ModelForm

class AuthorForm(ModelForm):
    class Meta:
        model = Author
        fields = '__all__'
```

Note: you don't need to create forms for next week's update.

Create views

- Show database table in HTML page
 - Import the model Class
 - Define a function and search the table for all objects
 - Add the resulting query set in the context
 - Pass the context in the render function
 - Also add the request and html page in the render function

StudentManagement/views.py

```
1 from django.shortcuts import render
2 from .models import Student
3 from .forms import StudentForm
4
5 # Create your views here.
6
7 def showStudents(request):
8     studentList = Student.objects.all()
9     context = {
10         'students': studentList
11     }
12     return render(request, 'StudentManagement/studentlist.html', context)
13
```



"You can't just copy-paste pseudocode into a program and expect it to work"



- Take input to database table from HTML page
 - Create an HTML page to take input. Just use form in the HTML page

```

1 {% extends 'base2.html' %}
2
3
4 {% block content %}
5
6 <form method="POST" enctype="multipart/form-data">
7     {% csrf_token %}
8     {{ form }}
9
10    <input type="submit">
11 </form>
12
13    <a href="{% url 'products_list' %}">Product List</a>
14
15
16 {% endblock content %}
17

```

- Create a views function to upload books

```

29 @login_required
30 def uploadProducts(request):
31     form = ProductForm()
32
33     if request.method == "POST":
34         form = ProductForm(request.POST, request.FILES)
35
36         if form.is_valid:
37             form.save()
38             return redirect('products_list')
39
40     context = {
41         'form': form
42     }
43
44     return render(request, 'ProductManagement/upload.html', context)
45

```

- Create an url for this function

```

21 from django.conf.urls.static import static
22
23
24 urlpatterns = [
25     path('admin/', admin.site.urls),
26     path('products/', product_views.showProducts, name='products_list'),
27     path('products/<int:product_id>', product_views.showDetails, name='detail_view'),
28
29     path('upload/', product_views.uploadProducts, name='upload_product'),
30
31     path('signup/', user_views.register, name='register'),
32
33     path('accounts/', include('django.contrib.auth.urls')),
34
35

```


When you only know
HTML and develop a
web application



Create HTML pages

1. Create directories with the APPs name
2. To show database table
 - a. Inside the template/app add a HTML page (example: showStudents.html)
 - b. Use a for loop to iterate through the query set
 - c. Print the information using {{ }}
3. To insert the data into database table
 - a. *Will be added later. Stay tuned!*

templates/StudentManagement/studentlist.html

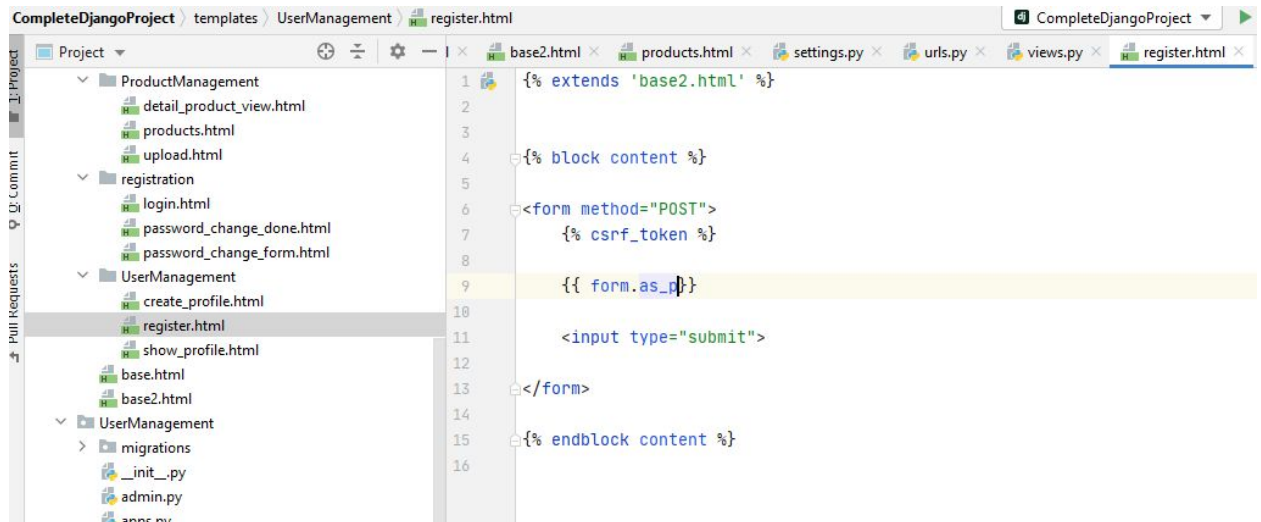
```
6      {% for s in students %}
7          <h2>Student info: {{ s.id }} </h2>
8          {{ s.name }} <br>
9          {{ s.email }} <br>
10         {{ s.contact_no }} <br>
11
12     {% endfor %}
```

When someone asks how is CSE 309 & 310 going ?



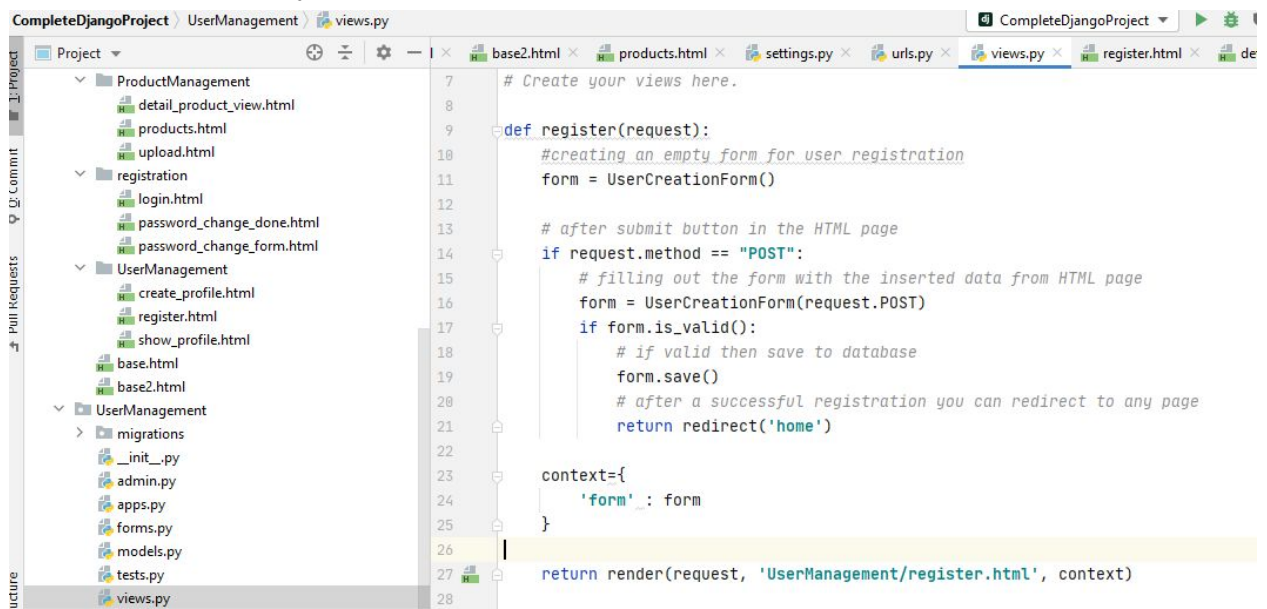
User Registration (Sign up)

1. Create a UserManagement App
2. Create an HTML page for user registration



```
1 {% extends 'base2.html' %}
2
3
4 {% block content %}
5
6 <form method="POST">
7     {% csrf_token %}
8
9     {{ form.as_p }}
10
11     <input type="submit">
12
13 </form>
14
15 {% endblock content %}
16
```

3. Write codes in views.py



```
1 # Create your views here.
2
3 def register(request):
4     #creating an empty form for user registration
5     form = UserCreationForm()
6
7     # after submit button in the HTML page
8     if request.method == "POST":
9         # filling out the form with the inserted data from HTML page
10        form = UserCreationForm(request.POST)
11        if form.is_valid():
12            # if valid then save to database
13            form.save()
14            # after a successful registration you can redirect to any page
15            return redirect('home')
16
17    context={
18        'form' : form
19    }
20
21    return render(request, 'UserManagement/register.html', context)
```

4. Create an url path and done!

```
path('signup/', user_views.register, name='register'),
```

Login

1. For authentication we will use the build in modules. Write this path in the urls.py

```
path('accounts/', include('django.contrib.auth.urls')) ,
```

Using the views

There are different methods to implement these views in your project. The easiest way is to include the provided URLconf in **django.contrib.auth.urls** in your own URLconf, for example:

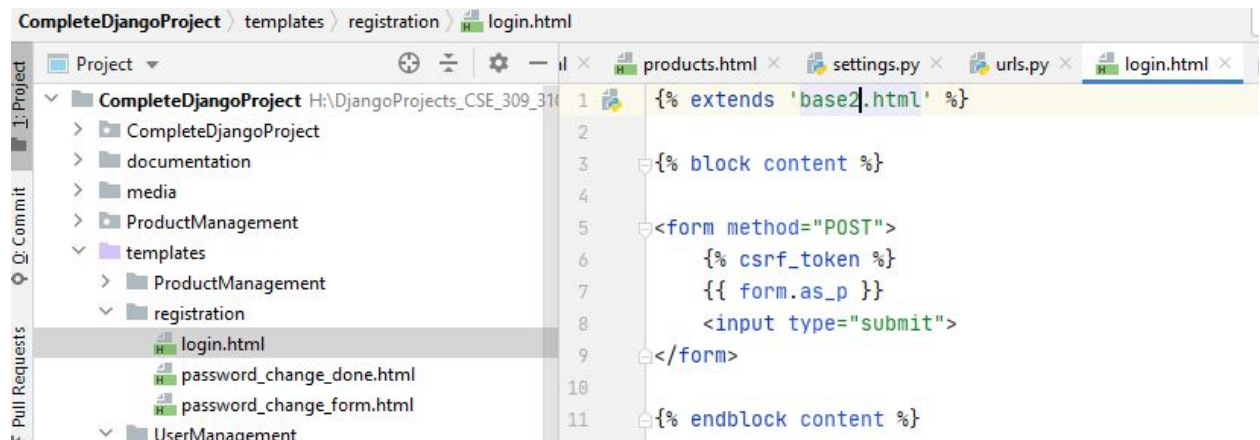
```
urlpatterns = [  
    path('accounts/', include('django.contrib.auth.urls')),  
]
```

This will include the following URL patterns:

```
accounts/login/ [name='login']  
accounts/logout/ [name='logout']  
accounts/password_change/ [name='password_change']  
accounts/password_change/done/ [name='password_change_done']  
accounts/password_reset/ [name='password_reset']  
accounts/password_reset/done/ [name='password_reset_done']  
accounts/reset/<uidb64>/<token>/ [name='password_reset_confirm']  
accounts/reset/done/ [name='password_reset_complete']
```

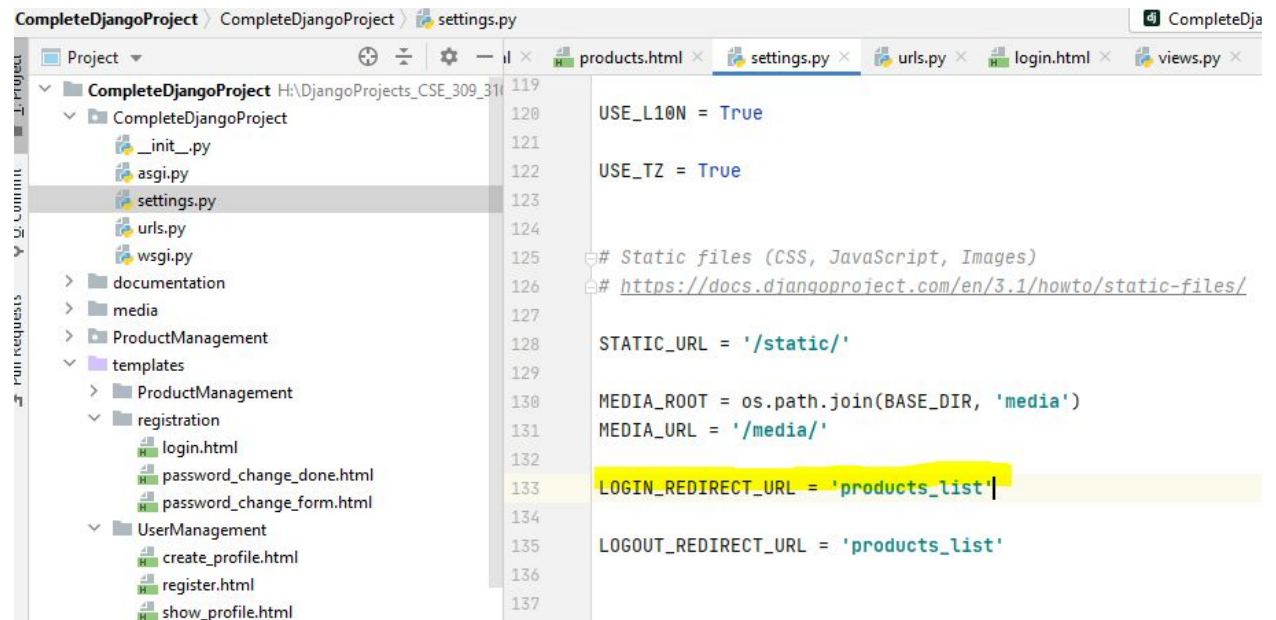
Ref: <https://docs.djangoproject.com/en/3.1/topics/auth/default/>

2. Create a “registration” directory in “templates” and add a “login.html” page.



3. You need to set Login route by adding a this line in the setting.py

LOGIN_REDIRECT_URL = 'your_url'



```
119
120 USE_L10N = True
121
122 USE_TZ = True
123
124
125 # Static files (CSS, JavaScript, Images)
126 # https://docs.djangoproject.com/en/3.1/howto/static-files/
127
128 STATIC_URL = '/static/'
129
130 MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
131 MEDIA_URL = '/media/'
132
133 LOGIN_REDIRECT_URL = 'products_list'
134
135 LOGOUT_REDIRECT_URL = 'products_list'
136
137
```

4. Now you can access the login page by using this url after the home url in browser

/accounts/login/

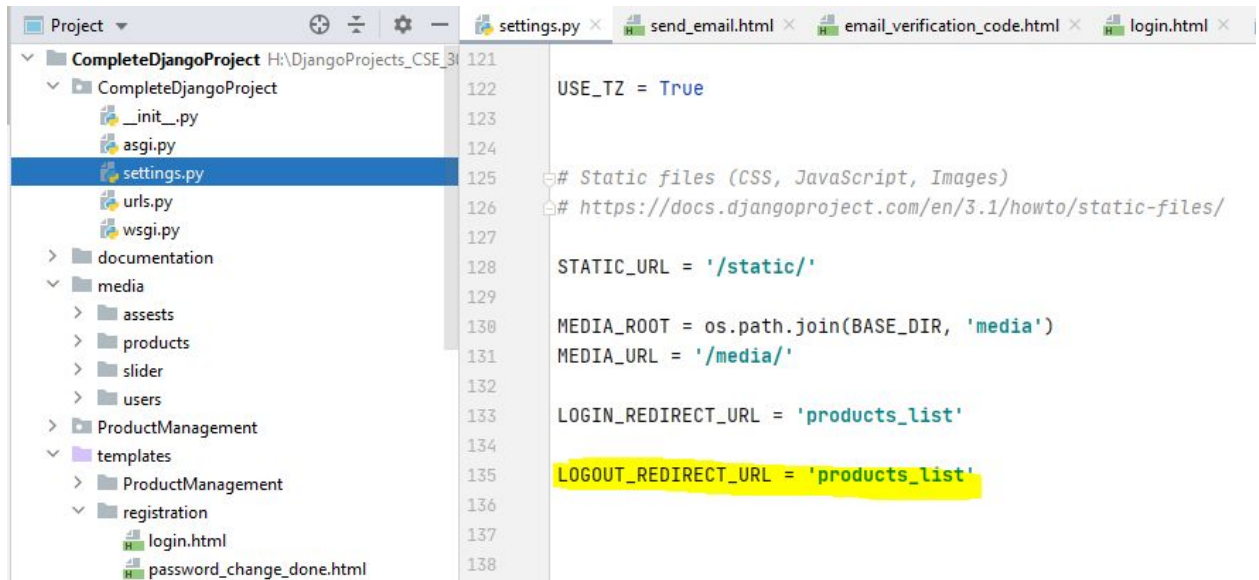
Python is the
easier language
to learn.
No brackets,
no main.

You get errors
for writing an
extra space



Logout

1. Just add `LOGOUT_REDIRECT_URL = 'your-url'` in `settings.py`



```
121
122 USE_TZ = True
123
124
125 # Static files (CSS, JavaScript, Images)
126 # https://docs.djangoproject.com/en/3.1/howto/static-files/
127
128 STATIC_URL = '/static/'
129
130 MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
131 MEDIA_URL = '/media/'
132
133 LOGIN_REDIRECT_URL = 'products_list'
134
135 LOGOUT_REDIRECT_URL = 'products_list'
136
137
138
```

2. Now you can access the logout function by using this url after the home url in browser

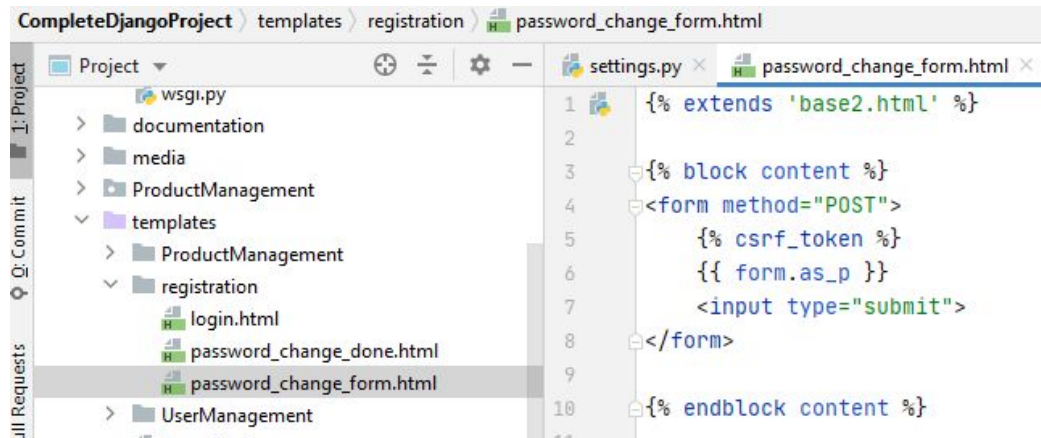
`/accounts/logout/`

3. Done!



Change Password

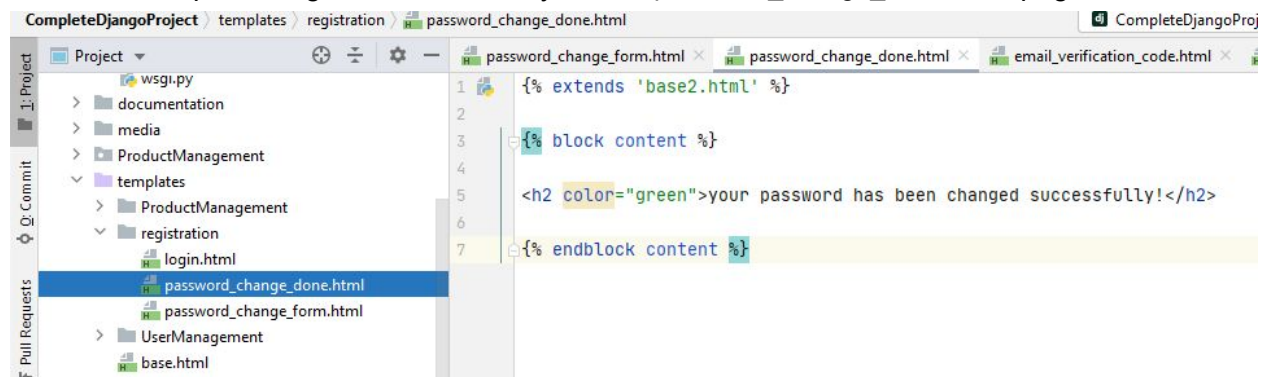
1. In “templates/registration” directory add a “password_change_form.htm” page



The screenshot shows the VS Code interface with the 'CompleteDjangoProject' workspace. The file explorer on the left shows the project structure, with 'templates/registration/password_change_form.html' selected. The main editor shows the content of 'password_change_form.html'.

```
1 {% extends 'base2.html' %}
2
3 {% block content %}
4     <form method="POST">
5         {% csrf_token %}
6         {{ form.as_p }}
7         <input type="submit">
8     </form>
9
10 {% endblock content %}
```

2. In “templates/registration” directory add a “password_change_done.html” page



The screenshot shows the VS Code interface with the 'CompleteDjangoProject' workspace. The file explorer on the left shows the project structure, with 'templates/registration/password_change_done.html' selected. The main editor shows the content of 'password_change_done.html'.

```
1 {% extends 'base2.html' %}
2
3 {% block content %}
4
5     <h2 color="green">your password has been changed successfully!</h2>
6
7 {% endblock content %}
```

3. Now you can access the password function by using this url after the home url in browser

/accounts/password_change/