Django Workshop

UNIVERSITY OF TORONTO
IEEE STUDENT BRANCH

Presented to you by - IEEE UofT Tech Team
October 12th 2022

What We Do

Software Workshops

Hackathons

UNIVERSITY OF TORONTO
IEEE STUDENT BRANCH

Hardware Workshops

Build Projects

Our Goal Today

To develop and host a To Do List App using Django.

By the end of this workshop you will:

- Be familiar with basic concepts used in Django.
- Learn how to use different components and involve them in an app.
- Build a personal app that could be useful in the upcoming midterm season. :D

Setting up VSCode

Under the assumption that you have installed VSCode on your machines, and followed the installation handout for Django.

Install Python, Django Extensions from the Extensions Section.

- Useful for keeping clean syntax.
- Autocomplete features.

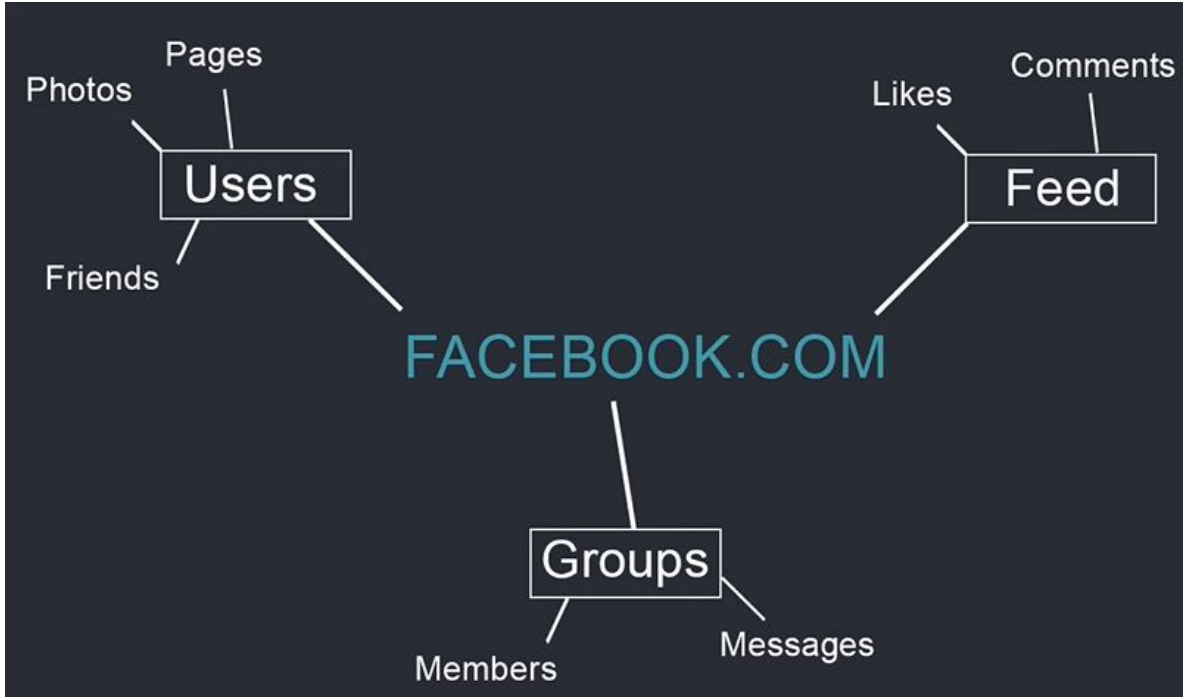
Basics of Django

Any website can be disintegrated into basic components called APPS.

APP1	APP2	APP3
Models Views URLs	Models Views URLs	Models Views URLs

Need to register these apps within the settings.py file of a project.

Example


Pages
Photos
Users
Friends
Comments
Likes
Feed
FACEBOOK.COM
Groups
Members
Messages

Views

- Functions that process a user's request when they visit a URL/endpoint on a website.
- When going to a website, the view associated is responsible for logic in backend; returns response in HTML Template/JSON data.
- Are of two types: function based and class based.

Views

```
class ProfileView(View):
    def get(self, request):
        user = getUser()
        return render(request, 'profile.html', {'user':user})
```

Models

- Models are representations of databases.
- Class represents the database table.
- Attributes within the Class represent columns within the table.
- Relations can be defined as one-to many, many-to-one, etc.

Models

```
class Project(models.Model):
    title = models.charField()
    description = models.TextField()
    id = models.UUIDField()
```

ID	TITLE	DESCRIPTION

Migration

- Django is designed to work with relational databases (MySQL, SQLite).
- Translates the Models we create into databases using ORM.
- Migration allows creation and modification of database tables without having to work with its native code.
    - You don’t have to manually create databases in MySQL.

URL Routing

- Routing is navigating between different parts of application.
- Handing routing in application by creating a list of urls in urls.py.

```
from django.urls import path
from . import views


urlpatterns = [
    path('profile/', views.profileView),
]
```

- First parameter is the URL, and second is the view to render when the URL is encountered.

Initial Setup

- Project is displayed on localhost:8000

  django-admin startproject todo_list

  django-admin startproject {name of project}

- I had to use ‘python -m django startproject todo_list’
- “python manage.py startapp {app name}”
  - (creates files for app, in this case app is named base)
- “python manage.py runserver”
  - (starts server, we can do our work with this server running and check our progress by refreshing the page)

Point to the app in Settings

- Add '{appname}.apps.BaseConfig' to your settings.py file
- This connects your app to Django

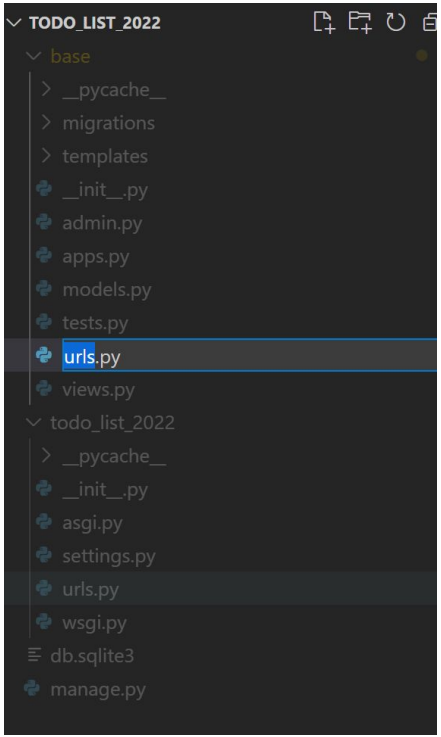
settings.py

todo_list_2022 > settings.py > ...

```python
# See https://docs.djangoproject.com/en/4.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-54t=^!t6(682m_e$1xy)pik7y#$4uz7@n$72lmldlhkgn&by&_'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'base.apps.BaseConfig'
]
```

Add a url file into App Folder

- Create a urls.py file in your APP folder NOT the PROJECT folder
- Setup url patterns within this file



Add a url file into App Folder

settings.py urls.py 2

base > urls.py > ...

```python
from django.urls import path
from .views import TaskDelete, TaskList, TaskDetail, TaskCreate, TaskUpdate, CustomLoginView, RegisterPage
from django.contrib.auth.views import LogoutView

urlpatterns = [
    path('login/', CustomLoginView.as_view(), name='login'),
    path('logout/', LogoutView.as_view(next_page='login'), name='logout'),
    path('register/', RegisterPage.as_view(), name='register'),
    path('', TaskList.as_view(), name='tasks'),
    path('task/<int:pk>/', TaskDetail.as_view(), name='task'),
    path('task-create/', TaskCreate.as_view(), name='task-create'),
    path('task-update/<int:pk>/', TaskUpdate.as_view(), name='task-update'),
    path('task-delete/<int:pk>/', TaskDelete.as_view(), name='task-delete'),

]
```

Add a url file into App Folder

- In the project urls.py file make sure to link {app name}.urls to Django

settings.py | urls.py base 2 | urls.py todo_list_2022 3

todo_list_2022 > urls.py > ...

```python
"""todo_list_2022 URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.1/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('base.urls')),
]
```

Setting up the Task Model

- This is the model we will use to store our task data
- Note the null and blank arguments allow the fields to be empty
- Meta class allows us to order the query set and move completed items to the bottom

models.py > ...

```python
from django.db import models
from django.contrib.auth.models import User
# Create your models here.

class Task(models.Model):
    # on delete can be changed to Set-null
    user = models.ForeignKey(User, on_delete=models.CASCADE, null=True, blank=True)
    title = models.CharField(max_length=200)
    description = models.TextField(null=True, blank=True)
    complete = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title

    class Meta:
        ordering = ['complete']
```

Migrating the Database

Run: python manage.py makemigrations

- Creates a file in the app folder
- The file runs SQL commands to create the table for the database
- Everytime the model is updated you will have to rerun the migration to update the database

Creating an admin user

Run: python manage.py createsuperuser

- Follow the prompts (you can use a fake email)

Goto: localhost:8000/admin

- Register the Task model in the admin.py file

base > admin.py

```python
from django.contrib import admin
from .models import Task
# Register your models here.

admin.site.register(Task)
```

Setting up Views

- Views for task list functionality
- These require html forms which will setup soon
- Make sure to import all this!

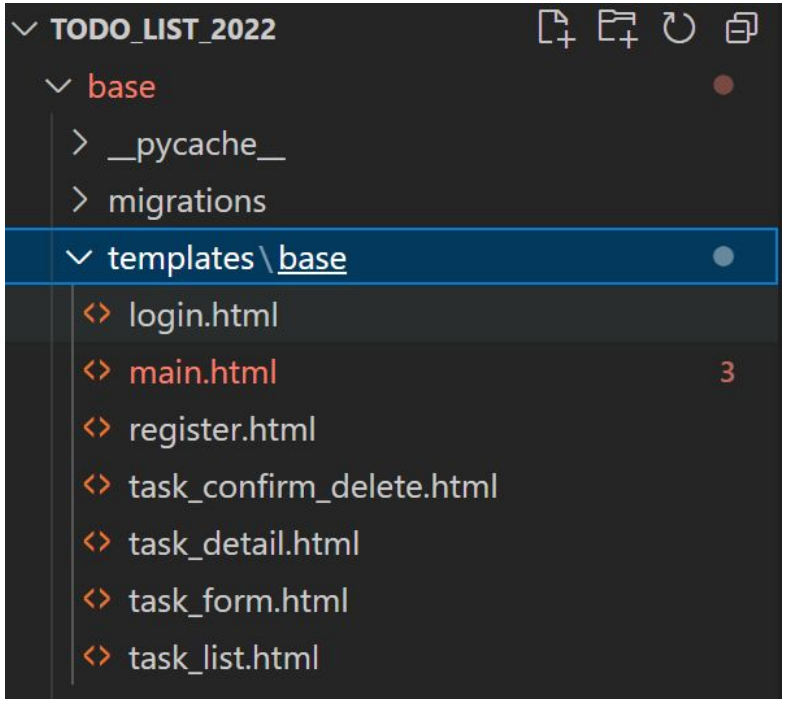
views.py > ...

```python
from django.views.generic.edit import CreateView, UpdateView, DeleteView, FormView
from django.urls import reverse_lazy
from django.shortcuts import render
from django.views.generic.list import ListView
from django.views.generic.detail import DetailView

from django.contrib.auth.views import LoginView
from django.contrib.auth.mixins import LoginRequiredMixin
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth import login
from .models import Task
```

```python
class TaskList(ListView):
    model=Task
    context_object_name = 'tasks'

class TaskDetail(DetailView):
    model = Task
    context_object_name = 'task'
    template_name = 'base/task.html'

class TaskCreate(CreateView):
    model = Task
    #list out all fields
    fields = '__all__'
    # Send user back to tasks page after creating a task
    success_url = reverse_lazy('tasks')

class TaskUpdate(UpdateView):
    model = Task
    fields = '__all__'
    # Send user back to tasks page after creating a task
    success_url = reverse_lazy('tasks')

class DeleteView(DeleteView):
    model = Task
    context_object_name = 'task'
    success_url = reverse_lazy('tasks')
```

Creating Templates

- In the app create a folder named 'templates'
- Create another folder within templates named '{app_name}'
- Create a file named 'task_list.html'
- Python Logic in HTML with Django by using
  - {% ... %}



task_list.html

- Main form for your to do list
- Notice the references to the views we made

```
<hr>
<h1>To Do List</h1>
<a href="{% url 'task-create' %}">Add Task</a>
<table>

    <tr>
        <th>Item</th>
        <th></th>
    </tr>
    {% for task in object_list %}
    <tr>
        <td>{{task.title}}</td>
        <!-- Links to tasks in Django -->
        <td><a href="{% url 'task' task.id %}">View</a></td>
        <td><a href="{% url 'task-update' task.id %}">Edit</a></td>
        <td><a href="{% url 'task-delete' task.id %}">Delete</a></td>
    </tr>
    {% empty %}
    <h3>No Items in List</h3>
    {% endfor %}
</table>
```

task_form.html

- This is how we will add tasks to the todo list with a POST request
- The csrf token is required for security
- The file must follow the _form.html syntax

task_form.html > form

```
<h3>Task Form</h3>
<!--  Back Button  -->
<a href="{% url 'tasks' %}">Go Back</a>
<form action="" method="POST">
    <!-- Token is for security -->
    {% csrf_token %}
    {{form.as_p}}
    <input type="submit" value="Submit">
</form>
```

task_confirm_delete.html

- This is how we will use our delete view.
- The file must follow the _confirm_delete.html syntax

task_confirm_delete.html > form

```
<a href="{% url 'tasks' %}">Go Back</a>
<form method="POST">
    {% csrf_token %}
    <p>Are you sure you want to delete this task? "{{task}}"</p>
    <input value="Delete" type="submit">
</form>
```

Setting up Login

- Add the following check to task_list.html
- Create the CustomLoginView
- LoginRequiredMixin allows the tasklist to be restricted to logged in users
- Add LOGIN_URL = “login” to settings.py to redirect login

no_style_task_list.html > hr

```
{% if request.user.is_authenticated %}
    <p>{{request.user}}</p>
    <a href="{% url 'logout' %}">Logout</a>
{% else %}
<a href="{% url 'login' %}">Login</a>
{% endif %}


<hr>
<h1>To Do List</h1>
<a href="{% url 'task-create' %}">Add Task</a>
```

todo_list_2022 > settings.py > LOGIN_URL

```
            'NAME': 'django.contrib.auth.password_validation.NumericPasswordVali
        },
    ]


    # Internationalization
    # https://docs.djangoproject.com/en/4.1/topics/i18n/

    LANGUAGE_CODE = 'en-us'

    TIME_ZONE = 'UTC'

    USE_I18N = True

    USE_TZ = True

    LOGIN_URL = 'login'
    # Static files (CSS, JavaScript, Images)
    # https://docs.djangoproject.com/en/4.1/howto/static-files/
```

```
class CustomLoginView(LoginView):
    template_name = 'base/login.html'
    fields = '__all__'
    redirect_autheticated_user = True

    # override success url
    def get_success_url(self):
        return reverse_lazy('tasks')


class TaskList(LoginRequiredMixin, ListView):
    model=Task
    context_object_name = 'tasks'
```

Login

- Add ‘LoginRequiredMixin’ to each view
- Define new method to make sure the logged in user only gets their tasks as well as a count of incomplete items

```
class TaskList(LoginRequiredMixin, ListView):
    model=Task
    context_object_name = 'tasks'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['tasks']= context['tasks'].filter(user=self.request.user)
        # count of incomplete items
        context['count']= context['tasks'].filter(complete=False).count()
        return context
```

Remove Dropdown to create task for other users

- Define new method to accept forms for the logged in user
- Change ‘fields’ from ‘__all__’ to a list of desired fields
    - [‘title’, ‘description’, ‘complete’]

```
class TaskCreate(CreateView):
    model = Task
    #list out all fields
    fields = ['title','description','complete']
    # Send user back to tasks page after creating a task
    success_url = reverse_lazy('tasks')

    # triggered by default on POST
    def form_valid(self,form):
        form.instance.user = self.request.user
        return super(TaskCreate, self).form_valid(form)


class TaskUpdate(UpdateView):
    model = Task
    fields = ['title','description','complete']
    # Send user back to tasks page after creating a task
    success_url = reverse_lazy('tasks')
```

Setting Up Register

- There is no register view built into Django so we will make a custom one
- This requires another html file!! (register.html)
- Also, update login.html to redirect to register

```python
class RegisterPage(FormView):
    template_name = 'base/register.html'
    form_class = UserCreationForm
    redirect_authenicated_user = True
    success_url = reverse_lazy('tasks')

    #redirect user once form is submitted
    def form_valid(self,form):
        user = form.save()
        if user is not None:
            login(self.request, user)
        return super(RegisterPage, self).form_valid(form)

    def get(self, *args, **kwargs):
        if self.request.user.is_authenticated:
            return redirect('tasks')
        return super(RegisterPage, self).get(*args, **kwargs)
```

```
login.html > p
<h1>Login</h1>

<form method="POST">
    {% csrf_token %}
    {{form.as_p}}
    <input type="submit" value="Login">
</form>

<p>Don't have an account? <a href="{% url 'register' %}">Register</a></p>
```

```
register.html > p > a
<h1>Register</h1>

<form method="POST">
    {% csrf_token %}
    {{form.as_p}}
    <input type="submit" value="Register">
</form>

<p>Already have an account? <a href="{% url 'login' %}">Login</a></p>
```

Search

- Add a form to task_list.html
- Implement the search filter into TaskList View

```python
class TaskList(LoginRequiredMixin, ListView):
    model=Task
    context_object_name = 'tasks'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['tasks']= context['tasks'].filter(user=self.request.user)
        # count of incomplete items
        context['count']= context['tasks'].filter(complete=False).count()

        search_input = self.request.get('search-area') or ''
        if search_input:
            context['tasks'] = context['tasks'].filter(title__icontains=search_input)

        context['search_input'] = search_input
        return context
```

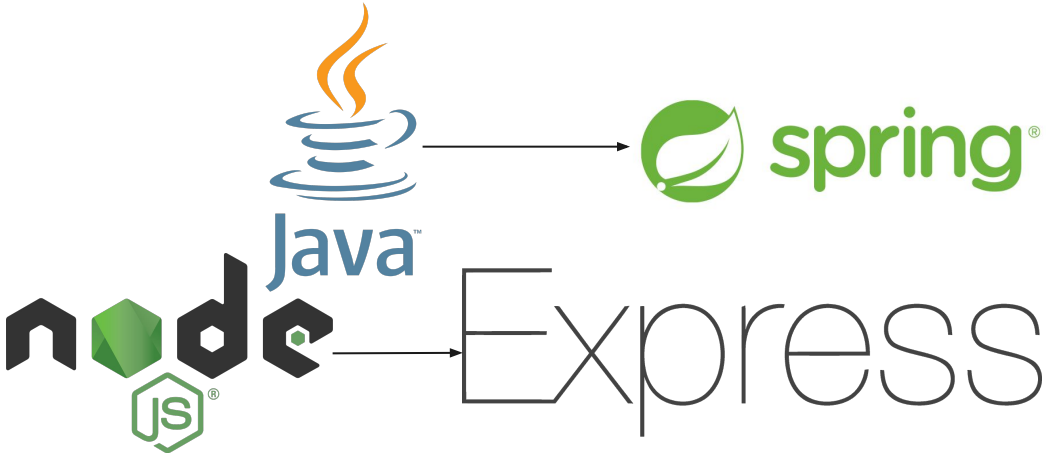
no_style_task_list.html > form

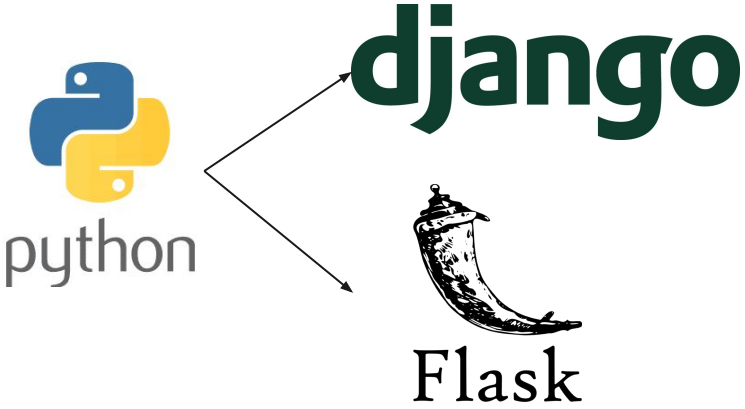
```html
{% if request.user.is_authenticated %}
    <p>{{request.user}}</p>
    <a href="{% url 'logout' %}">Logout</a>
{% else %}
<a href="{% url 'login' %}">Login</a>
{% endif %}


<hr>
<h1>To Do List</h1>
<a href="{% url 'task-create' %}">Add Task</a>


<form method="GET">
    <input type="text" name="search-area" value="{{search_input}}">
    <input type="submit" value="Search">

</form>
<table>
```

With some CSS styling...

Web Frameworks

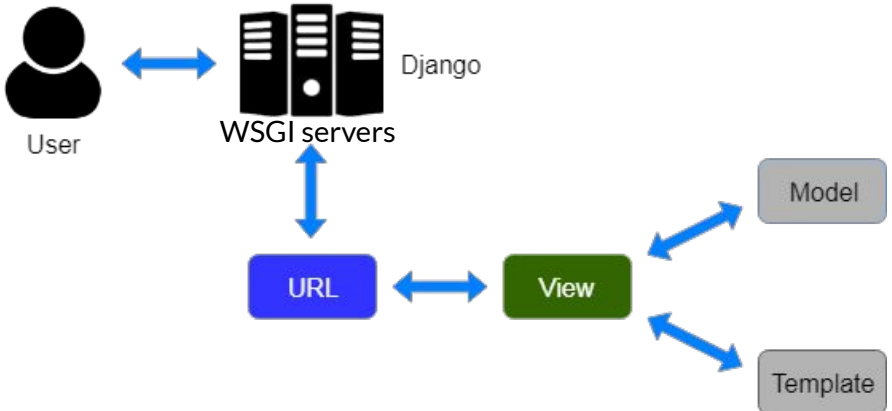
- Core Python is not *enough*.
- Web Frameworks are frameworks for languages that support the development and deployment of web applications
- Automates the overhead associated with common activities performed in web development.





What is  ?

- Django is a Python-based web framework that follows the model–template–views architectural pattern.
- Allows you to create full-stack web applications without reinventing the wheel.
- “Batteries Included”: authentication, HTTP libraries, template engine, object-relational mapper (ORM), etc.
- Rapid Development, Secure, Versatile, Open Source, Supported Community.
- Companies that use Django
  - Instagram
  - Mozilla
  - Pinterest
  - Bitbucket
  - The Washington Times



 vs the others: Objective Review

-  Flask, a micro framework, easy to pick up and build basic apps.
- Express is less complex, more flexible and can be used with React.
-  is JDBC based, and also highly flexible.

Which one to go with? Choose the one that supports your favorite language!