Week 9 Web Application Architecture and Design

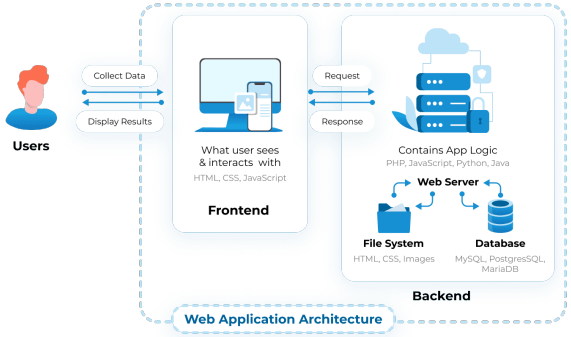
Dr Zhi Zhang

Overview

- Web Application Architecture
- Web Application Design
- Django Framework

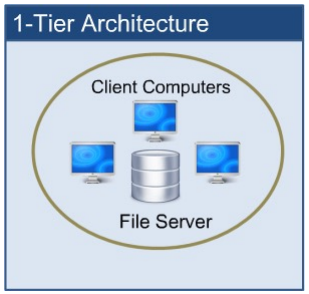
What is a web application architecture

- A web application architecture is a layout that displays the interactions between different software components, such as frontend, and backend.



https://litslink.com/blog/web-application-architecture#types-of-web-application-architecture

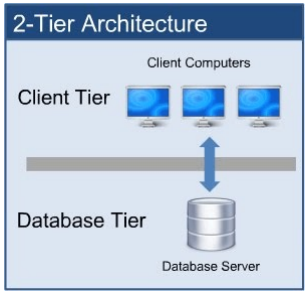
1-Tier web application architecture



1-Tier Architecture/Monolithic Architecture: all the software components are available on the same machine.

https://www.perfmatrix.com/software-architecture-and-its-types/

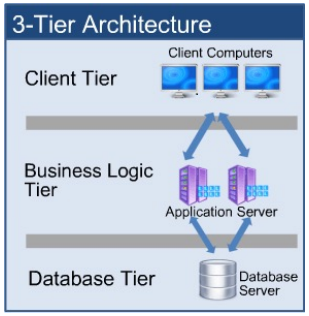
2-Tier web application architecture



2-Tier Architecture/Client-Server Architecture: The client sends the request to the server and the server system processes the request and sends the response back to the client.

https://www.perfmatrix.com/software-architecture-and-its-types/

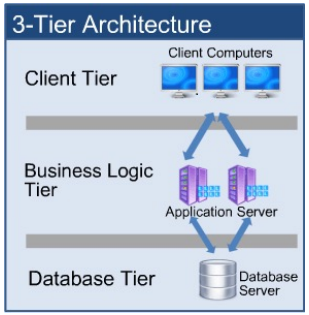
3-Tier web application architecture



3-Tier Architecture: a common web application architecture. The intermediate application server receives client requests and processes them by applying the business logic. The communication between the client and the database is managed by the intermediate application layer.

https://www.perfmatrix.com/software-architecture-and-its-types/

3-Tier web application architecture



Presentation Layer: displays the user interface and manages user interaction.

Application Layer/Business Layer: has all the business logic, rules and policies.

Data Layer: stores and maintains the data.

https://www.perfmatrix.com/software-architecture-and-its-types/

Distributed web application architecture

Presentation Layer**:** displays the user interface and manages user interaction.

Application Layer/Business Layer: has all the business logic, rules and policies.

Data Layer: stores and maintains the data.

Layers of different services: defines separate components or modules that perform specific tasks or provide functionality to the overall web application.

Examples: Caching service, Job Server, Full-Text Search Service, Datawarehouse.

Web application architecture

- It is the skeleton: outlining how different software components are organized, and interact with each other.

Web application design

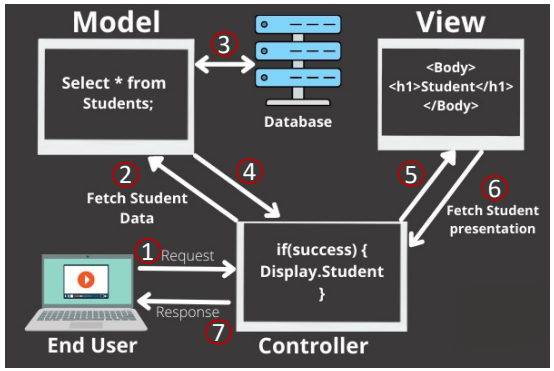
- It is the code-level design of the components and their interaction. It is the level of modules, functions, classes, interfaces, etc, which are used to implement each component and the interaction.

A popular design pattern

- Model—View—Controller (MVC):
  - **Model:** maintains the application data, e.g., interacts with the database.
  - **View:** provides templates for visualizing the application data retrieved from the model.
  - **Controller:** acts as an intermediary between the Model and the View, e.g., reads/writes data via the Model component, and interacts with the View component to render the output.

Model—View—Controller (MVC)

- An example:



https://www.geeksforgeeks.org/mvc-framework-introduction/

A short summary

- Model—View—Controller:
  - **Model:** interacts with data.
  - **View:** visualizes data.
  - **Controller**: tells the model and view of what to do.

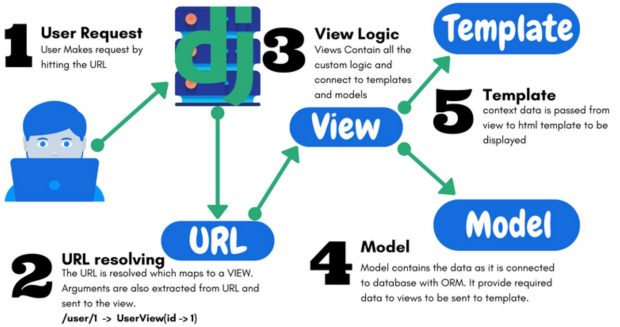
Django

- Django is an open-source web application framework written in Python.
  - Django customizes the MVC design pattern.

- Django customizes MVC as MVT (Model—View—Template)
  - M — — — ->For data interaction— — — — — ->M
    V — — — -> For data visualization— — — — — ->T
    C — — — -> For component interaction— — — —>V

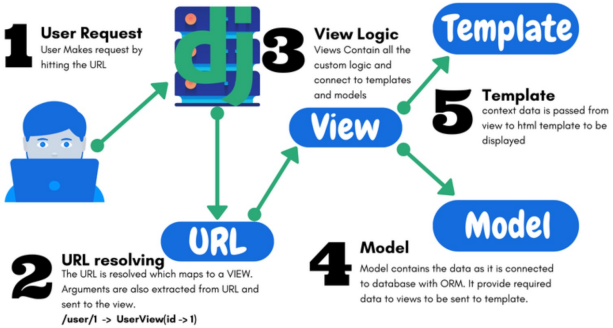
- Any real-world Django-based web application?
  - Instagram, Spotify, Youtube

- A Django-based web application:



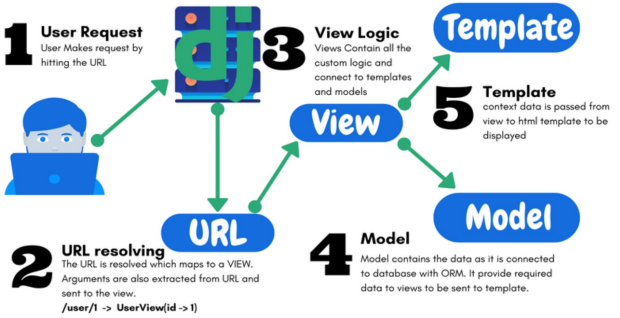
- **URL**: resolves a request from a given URL
  - Extracts specific patterns of strings and/or digits in the URL and sends them to the VIEW function.

- A Django-based web application:



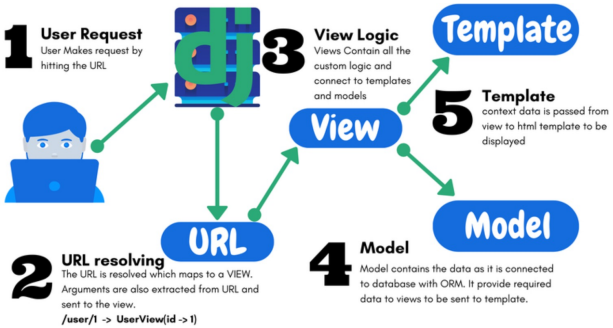
- **View**: it is an HTTP request handler function that receives and responds to HTTP requests.
  - Interacts with templates and models as needed.

- A Django-based web application:



- **Model:** interacts with the application data in a database, e.g., creating a database table.

- A Django-based web application:



- **Template**: defines the structure or layout of web pages, with placeholders for actual content.

- Sending a request to a Django web application: example code
- urls.py: implements the URL component

```
from django.urls import path
from django.contrib import admin
from music import views

urlpatterns  =  [
      path('', views.home, name='home'),
      path('admin/', admin.site.urls),
]
```

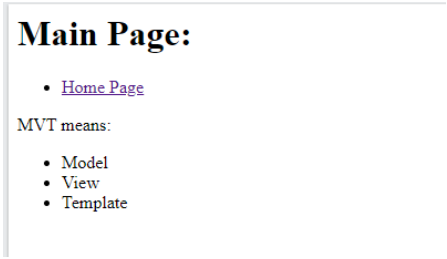
- Handling the request: example code
- views.py: implements the View component

```
from django.shortcuts import render, redirect
from django.http import HttpResponse

# Create our views here.
def home(request):
      return render(request, 'home.html')
```

- html template for data rendering: example code

```
<h1>Main Page:</h1>
<ul>
  <li><a href="{% url 'home' %}">Home Page</a></li>
</ul>
<p>MVT means:</p>
<ul>
  <li>Model</li>
  <li>View</li>
  <li>Template</li>
</ul>
```

- html template for data rendering: example code



- Managing the application data: example code
- models.py: implements the Model component

```
1  from django.db import models
2  # Define your models here.
3
4  class Artist(models.Model):
5        name = models.CharField(max_length=100)
6        song = models.CharField(max_length=25)
```

Example: Create a Django web application called music

Step 1: Setting up a virtual environment

```
mkdir ~/cits5503_django
cd cits5503_django
python3 -m venv cits5503venv
source cits5503venv/bin/activate
```

Question: /opt/wwc/ is not a good path for deploying a Django application:
While this path requires the root privilege, the application itself does not require it.

Step 1: Setting up a development environment

```
mkdir ~/cits5503_django
cd cits5503_Django
python3 -m venv cits5503venv
source cits5503venv/bin/activate
```

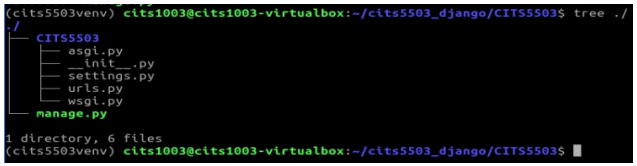
Question: Why do we need to setup a virtual environment in python?

Isolation: Virtual environments provide a way to isolate the python package dependencies of different Python projects.

Ease of Deployment: Virtual environments make it easier to deploy a python project to different systems. All the installed packages in a virtual environment can be copied onto another machine or virtual environment.

Step 2: Setting up a Django project and app

```
pip install django
django-admin startproject CITS5503
cd CITS5503
```




```
python3 manage.py startapp music
```

Step 2: Setting up a Django project and app

```
pip install django
django-admin startproject CITS5503
cd CITS5503
python3 manage.py startapp music
```

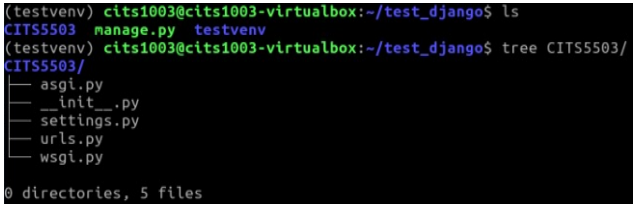
Question: What's the difference between the outputs of

django-admin startproject CITS5503

django-admin startproject CITS5503 .

```
django-admin startproject CITS5503
```


```
django-admin startproject CITS5503 .
```



Step 2: Setting up a Django project and app

```
pip install django
django-admin startproject CITS5503
cd CITS5503
python3 manage.py startapp music
```

Question: What's the difference between the outputs of

django-admin startproject CITS5503: Django creates a new directory named "CITS5503" , inside which another subdirectory named "CITS5503" is created: the python project files is placed inside the subdirectory.

django-admin startproject CITS5503 . : Django creates a new directory named "CITS5503".

Step 3: Starting the Django application server

```
python manage.py check
python manage.py runserver
```

Question: Which default port is the Django application server is listening on?

Port: 8000

Question: What if the port is taken by another process in our system?

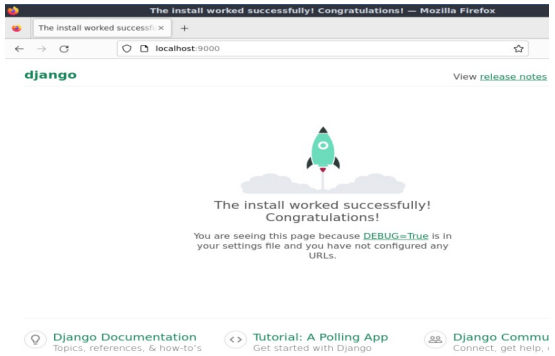
Step 3: Starting the Django application server

Question: What if the port is taken by another process in our system?

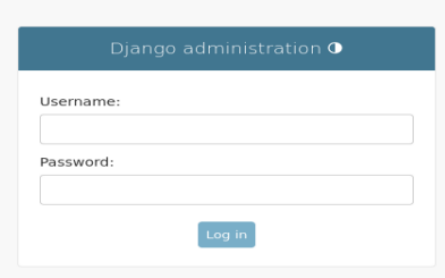
```
python manage.py runserver 9000
```

```
#ngnix

proxy_pass http://127.0.0.1:8000;
```

Open a browser and go to: http://localhost:9000



If we go to: http://localhost:9000/admin/



Step 4: Setting up an admin

```
python manage.py migrate
```

```
(cits5503venv) cits1003@cits1003-virtualbox:~/cits5503_django/CITS5503$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

Step 4: Setting up an admin

```
python manage.py migrate
```

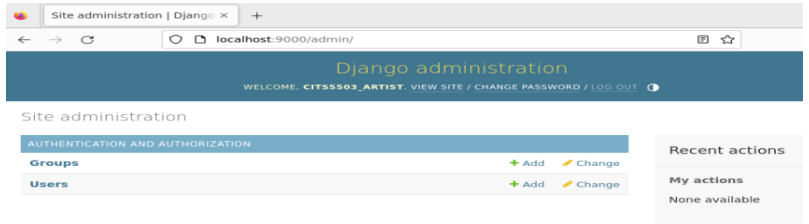
```
(cits5503venv) cits1003@cits1003-virtualbox:~/cits5503_django/CITS5503$ ls
CITS5503  db.sqlite3  manage.py  music  templates
```

```
(cits5503venv) cits1003@cits1003-virtualbox:~/cits5503_django/CITS5503$ sqlite3 db.sqlite3
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for usage hints.
sqlite> .tables
auth_group                  django_admin_log
auth_group_permissions      django_content_type
auth_permission             django_migrations
auth_user                   django_session
auth_user_groups            music_artist
auth_user_user_permissions
```

Step 4: Setting up an admin

```
python manage.py createsuperuser
```

```
(cits5503venv) cits1003@cits1003-virtualbox:~/cits5503_django/CITS5503$ python manage.py createsuperuser
Username (leave blank to use 'cits1003'): cits5503_artist
Email address: zhi.zhang@uwa.edu.au
Password:
```



Step 5: Setting up the music app

```
# inside CITS5503/settings.py

INSTALLED_APPS = [
      'django.contrib.admin',         #The admin site
      'django.contrib.auth',          #Authentication
      'django.contrib.contenttypes', #A framework for content types
      'django.contrib.sessions',      #A session framework
      'django.contrib.messages',      #A messaging framework
      'django.contrib.staticfiles',  #A framework for managing static files

      #new app below:
      'music.apps.MusicConfig',
]
```

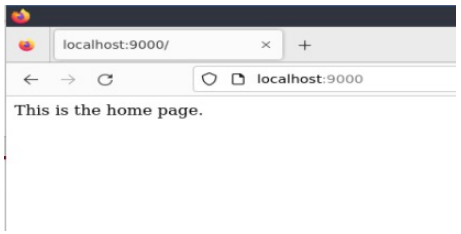
Step 6: Updating the url component in the project

```
# inside CITS5503/urls.py

from django.contrib import admin
from django.urls import path
from music import views # added

urlpatterns = [
     path('', views.home, name='home'), # added
     path('admin/', admin.site.urls),
]
```

Step 7: Updating the view component in the app

```
# inside music/views.py

from django.shortcuts import render
from django.http import HttpResponse #added

#added
def home(request):
     return HttpResponse('This is the home page.')
```

go to: http://localhost:9000



Step 8: Creating html templates in the project and the app

Inside the current working directory of the project

```
mkdir templates
mkdir music/templates
mkdir music/templates/music
```

```
touch templates/home.html
touch music/templates/music/main.html
touch music/templates/music/artist.html
```

templates/home.html

```
<h1>Home Page:</h1>
<ul>
<li><a href="{% url 'home' %}">Home Page</a></li>
<li><a href="{% url 'music:main' %}">Music Main Page</a></li>
<li><a href="{% url 'music:artist' %}">Artist Page</a></li>
</ul>
```

Home Page:

- Home Page
- Music Main Page
- Artist Page

music/templates/music/main.html

```
<h1>Music Main Page:</h1>
<ul>
<li><a href="{% url 'home' %}">Home Page</a></li>
<li><a href="{% url 'music:main' %}">Music Main Page</a></li>
<li><a href="{% url 'music:artist' %}"> Artist Page </a></li>
</ul>
```

Music Main Page:

- Home Page
- Music Main Page
- Artist Page

music/templates/music/artist.html

```
<h1>Artist Page:</h1>
<p>name: Jerry</p>
<p>song: Call me today</p>
<ul>
<li><a href="{% url 'home' %}">Home Page</a></li>
<li><a href="{% url 'music:main' %}">Music Main Page</a></li>
<li><a href="{% url 'music:artist' %}"> Artist Page </a></li>
</ul>
```

Artist Page:

name: Jerry

song: Call me today.

- Home Page
- Music Main Page
- Artist Page

Step 9: Updating the url component in the project

```
# inside CITS5503/urls.py

from django.contrib import admin
from django.urls import path
from music import views # added

urlpatterns = [
     path('', views.home, name='home'), # added
     path('admin/', admin.site.urls),
     path('music/', include('music.urls')), # added
]
```

Step 10: Creating the url component in the app

```
touch music/urls.py
```

```
# inside music/urls.py
from django.urls import path
from . import views
app_name = 'music'
urlpatterns = [
     path('', views.main, name='main'),
     path('artist/', views.artist, name='artist'),
]
```

Question: What is the url pattern for the first path?

localhost:9000/music

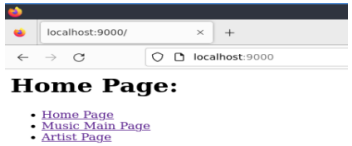
Step 11: Updating the view component in the app

```
# inside music/views.py

from django.shortcuts import render

#updated
def home(request):
     return render(request, 'home.html')
def main(request):
     return render(request, 'music/main.html')
def artist(request):
     return render(request, 'music/artist.html')
```

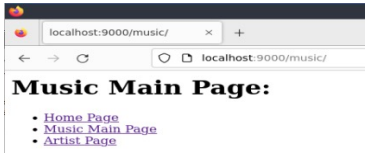
Step 12: Specifying the templates in the project

```
# inside CITS5503/settings.py

import os #added

TEMPLATES = [
{
(...)
      'DIRS': [os.path.join(BASE_DIR, 'templates')], #added
(...)
}
]
```

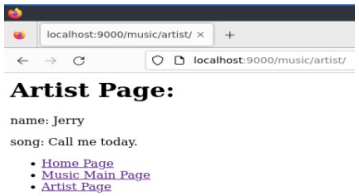
Go to the home page



Go to the main page



Go to the artist page



Automatically generated data

Step 13: Updating the model component in the app

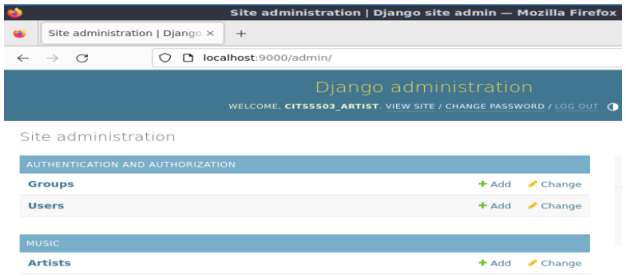
```
# inside music/models.py

from django.db import models

class Artist(models.Model):
     name = models.CharField(max_length=200)
     song = models.CharField(max_length=200)
     def __str__(self):
          return self.name
```

Step 14: Creating the Artist table and registering it into the admin interface

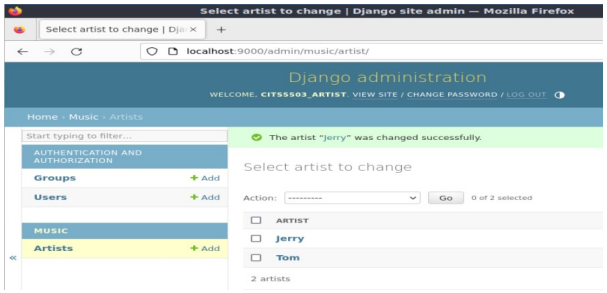
```
# inside music/admin.py

from django.contrib import admin
from .models import Artist

admin.site.register(Artist)
```

```
python manage.py makemigrations
python manage.py migrate
```

Go to the admin interface



Step 15: Populating the artist table



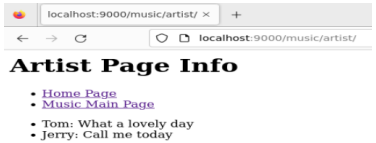

```
    def __str__(self):
        return self.name
```

Step 16: Updating the view component in the app

```
# inside music/views.py

from django.shortcuts import render
from .models import Artist #added

def home(request):
    return render(request, 'home.html')
def main(request):
    return render(request, 'music/main.html')
#updated
def artist(request):
    title = 'Artist Page Info'
    artist_list = Artist.objects.all()
    context = {'title': title, 'artist_list': artist_list}
    return render(request, 'music/artist.html', context)
```

Step 17: Updating music/templates/music/artist.html

```
<h1>{{title}}</h1>
<ul>
<li><a href="{% url 'home' %}">Home Page</a></li>
<li><a href="{% url 'music:main' %}">Music Main Page</a></li>
</ul>

<ul>
    {% for artist in artist_list %}
    <li> {{artist.name}}: {{artist.song}}</li>
    {% endfor %}
</ul>
```



Question: What if 'title' in context is replaced by 'titlectx'?

```
def artist(request):
      title = 'Artist Page Info'
      artist_list = Artist.objects.all()
      context = {'title': title, 'artist_list': artist_list}
      return render(request, 'music/artist.html', context)
```

```
def artist(request):
      title = 'Artist Page Info'
      artist_list = Artist.objects.all()
      context = {'titlectx': title, 'artist_list': artist_list}
      return render(request, 'music/artist.html', context)
```

```
<h1>{{titlectx}}</h1>
<ul>
<li><a href="{% url 'home' %}">Home Page</a></li>
<li><a href="{% url 'music:main' %}">Music Main Page</a></li>
</ul>

<ul>
      {% for artist in artist_list %}
      <li> {{artist.name}}: {{artist.song}}</li>
      {% endfor %}
</ul>
```

Otherwise