In Reference to Linkedin Learning Course Learning Django (2018)

https://www.linkedin.com/learning-login/share?forceAccount=false&redirect=https%3A%2F%2Fwww.linkedin.com%2Flearning%2Flearning-django-2018%3Ftrk%3Dshare_ent_url&account=56973065

Django Python

- It is a high-level web framework that encourages rapid development in a clean, pragmatic design.
- Web framework is a collection of tools for web development. Django has the following tools:
    - Object- Relational mapper
    - URL routing
    - Html templating
    - Form handling
    - Testing
- Django is not a programming language and it is also not a service.
- Django comes with this pytz library with supports time zone.
- ## Reference: https://www.linkedin.com/learning/learning-django/install-python-and-django?u=56973065

Create and run a minimal Django app

In Django terminology, a "Django project" is composed of several site-level configuration files along with one or more "apps" that you deploy to a web host to create a full web application. A Django project can contain multiple apps, each of which typically has an independent function in the project, and the same app can be in multiple Django projects. An app, for its part, is just a Python package that follows certain conventions that Django expects.

To create a minimal Django app, then, it's necessary to first create the Django project to serve as the container for the app, then create the app itself. For both purposes, you use the Django administrative utility, django-admin, which is installed when you install the Django package.

Create the Django project

- In the VS Code Terminal where your virtual environment is activated, run the following command:
- django-admin startproject web_project .
- This startproject command assumes (by use of . at the end) that the current folder is your project folder, and creates the following within it:
- manage.py: The Django command-line administrative utility for the project. You run administrative commands for the project using python manage.py <command> [options].
- A subfolder named web_project, which contains the following files:
- __init__.py: an empty file that tells Python that this folder is a Python package. It says that this contains python files
- wsgi.py: an entry point for WSGI-compatible web servers to serve your project. You typically leave this file as-is as it provides the hooks for production web servers. Provides a hook for web server. Such as Apache or Nginx.
- settings.py: contains settings for Django project, which you modify in the course of developing a web app. Configures Django.

- urls.py: contains a table of contents for the Django project, which you also modify in the course of development. Routes requests based on url.

##Reference: https://code.visualstudio.com/docs/python/tutorial-django

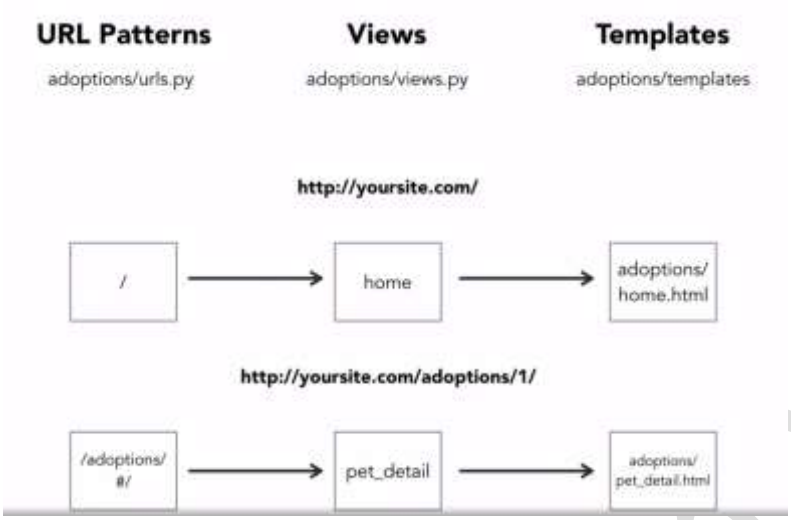
Django Apps

- An app is a folder with Python files.
- When you run Django app for the first time it creates a sql file.
- Command for creating an app is python manage.py startapp adoptions.
- Where adoptions is the app name.
- Startapp generates a folder in the name of the application and also creates something called a migration folder that we will discuss more in future.
- Then add the application that you just created into the installed application section in the setting.py file.

File / Folder	Role
apps.py	Configuration and initialization
models.py	Data layer
admin.py	Administrative interface
urls.py	URL routing
views.py	Control layer
tests.py	Tests the app
migrations/	Holds migration files

Models, Routing, Views, and Templates

- It uses a MVC (Model view control architecture)
  - url patterns
  - Views
  - Templates
  - Models
- Request is handled by url patterns
- Views provides the logic or controllable. It is a python callable such as a function. So it takes Http request and gives Http response.
- To perform queries against the database view can use Django model as needed.
- In is a place where you define the schema or underlying structure of a database table.
- Templates are also can be used by view for the presentation of the page to be defined.



- So, if a request is made for the website at '/' then route it to the home page function in the view.
- If you see the next example, in this we can potentially pass a variable as a request. This variable can then be queried into the database to retrieve relevant information.
- This combined with the template will give you something of a dynamic page.

Models

- Create the data layer of an app.
- Define database structure.
- Allow us to query the database.
- A model is a class inheriting from Django.db.models.Model, and is used to define fields as class attributes.
- As a rough analogy we can say that models is a spread sheet.
- Each model is a table.
- Each field is a column in the spreadsheet table.
- Record is a row.

```
from django.db import models

class Item(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    amount = models.IntegerField()
```

- Field type for numeric data
  - integerField
  - Decimal Field
- Field for textual data
  - Char Field
  - Text Field
  - Email Field
  - URL Field

- Miscellaneous Data Field
    - Boolean Field
    - DateTimeField
- Relational Data Field
    - Foreign Key
    - Many to Many
- Field Attribute Options
    - Max_length
    - Null
    - Blank
    - Default
    - Choices

Migrations

- Generates the script to change the database structure
- When a new model is defined, the initial migration will create the corresponding database tables.
- Migration is needed when:
    - Adding a Model
    - Adding a Field
    - Removing a Field
    - Changing a Field
- The command for migrations are:
    - Python manage.py makemigrations
        - Generates migration files for later use
        - Uses current fields and current database tables
        - Creates numbered files in appname/migrations/
    - Python manage.py migrate
        - Runs all migrations that haven't been run yet
        - Can also run migrations for an app to a specific number using:
          Migrate <appname> <number>
          Eg.
          Migrate adoptions 1
    - Python manage.py showmigrations
    - Unapplied Migrations
        - When a migration has been created, but hasn't been run
        - Very common source of error in development

Importing the data from csv file

- Even if the data is not exactly in the same format as in the database, we can load it by running a script that specifies exactly how to load the data.
- For this a management command is used with was out of the scope of this tutorial.
- But even-then looking at the program in the management/commands/loadpet_data.py
- We import some files like
    - csv(Direct Reader)
    - datetime for datetime format
    - Django.core.management import BaseCommand

  - Then we import models that we have generated for the database
  - Pytz import UTC
- After we have such a management script
- We use the command
  - Python3 manage.py load_pet_data
    As the management script name was that.

Working with the Django Admin

- Go to the admin file in the application.
- Import the model
-

```
from django.contrib import admin

from .models import Pet

@admin.register(Pet)
class PetAdmin(admin.ModelAdmin):
    list_display = ['name', 'species', 'breed', 'age', 'sex']
```

- Then to get the vaccination name instead of vaccine1 and so on.
- In this we are giving a decorator argument called register(Which takes an models as attribute).
- Then we create a class that inherits the admin.ModelAdmin.
- This class can take different attributes and then it can override many different method.
- Then we create a super user.
  - Python3 manage.py createsuperuser
  - Then give it admin name
  - Email is optional
  - And then password
  - The app is re run whenever you save changes

```python
from django.db import models

class Pet(models.Model):
    SEX_CHOICES = [('M', 'Male'), ('F', 'Female')]
    name = models.CharField(max_length=100)
    submitter = models.CharField(max_length=100)
    species = models.CharField(max_length=30)
    breed = models.CharField(max_length=30, blank=True)
    description = models.TextField()
    sex = models.CharField(choices=SEX_CHOICES, max_length=1, blank=True)
    submission_date = models.DateTimeField()
    age = models.IntegerField(null=True)
    vaccinations = models.ManyToManyField('Vaccine', blank=True)

class Vaccine(models.Model):
    name = models.CharField(max_length=50)

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

- Add the return name in the override method.
- This is in the models file where the database tables are formed.

Query data with the Django ORM

I did some of the basic command like get()

But there are a lot of other commands

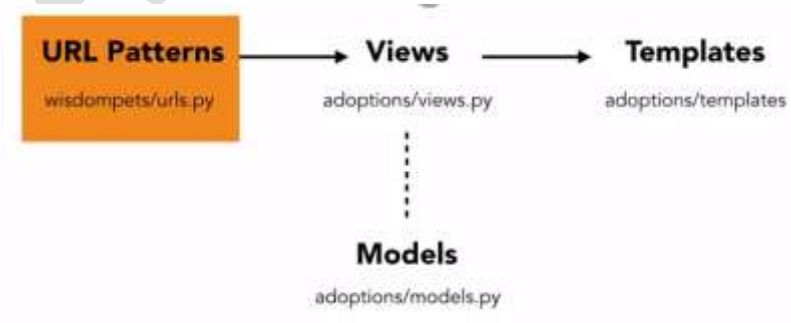
I used the shell for doing this all which can be opened by

Python manage.py shell

Building URL Handlers and Views

Understanding URL patterns

- At high level url patterns dispatch request to the view based on the path.



Regular Expressions

Regular	String That Matches
ducky	rubber ducky
\d	1
\d+	12 ounces
^admin/	admin/inventory/item/
suffix$	anything-suffix
^$	

- To check out more on the regular expression got to google.
- Also an important link is pythex.org

```
from django.confs.urls import url
from adoptions import views

urlpatterns = [
    url(r'^$', views.home, name='index'),
]
```

- R is for raw and it means that it will ignore the back slash.
- So this means if someone comes to the root that is "/" then they are taken to the home view.
- Which then will take the adoptions/home.html template.
- It also take name parameter which is also used in templates.
- If not match go to the other one.
- If no match found at all then go to home.

Implementing URL patterns

- For this we go to the urls.py file.
- There is a big comment that you can remove.
- For using url function you need to import url from Django.conf.urls.
- Then code something like this and for more specific information go to google and u can also try to check if they are working on pythex.org.

```
from adoptions import views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', views.home, name='home'),
    url(r'^adoptions/(\d+)/', views.pet_detail, name='pet_detail'),
]
```

- As we are using some views that we have not defined yet lets work on that.
- As the request are forwarded to the views that can then use templates.

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

def home(request):
    return HttpResponse('<p>home view</p>')

def pet_detail(request, id):
    return HttpResponse('<p>pet_detail view with the id {}</p>'.format(id))
```

- Now you can run the server to check if everything is working fine.

Implementing Django views

- Import models to query into it.
- Then we return a call to render function that takes (request , then template name and then, To pass data to the template to display the data we pass a dictionary to the function which needs to be string).
- See the code for better understanding.
- If we query something specific and there is a chance that an error can be thrown then address it by a try except block.

```
from django.shortcuts import render
from django.http import HttpResponse
from django.http import Http404

from .models import Pet

def home(request):
    pets = Pet.objects.all()
    return render(request, 'home.html', {'pets': pets})

def pet_detail(request, id):
    try:
        pet = Pet.objects.get(id=id)
    except Pet.DoesNotExist:
        raise Http404('Pet not found')
    return render(request, 'pet_detail.html', {'pet': pet})
```

- Then we need to create an template folder inside the application folder.
- This is the place where all the templates are going to be created. These templates are html files and are going to be explained further.
- The name of the template also includes the extension.

Building Django Templates

Django Templates

- {{variable}} needs to be in double curly brace.

- Then there is template tag which is inside a {% tag %}.
- These are used for loops, if, structural elements and control logic.
- {{ variable|filter }} pip for filter
    - Data time formatting
    - Forcing text into title or upper case

Other important topics:

- Implement Django Templates
- Structure templates
- Integration CSS and Javascript