**Notes for Django framework:**

**- Creating first project:**

**-------------------------**

- Create starter files that every app has: django-admin startproject [name of project]- creates a directory, with the name.

(ex: django-admin startproject [name\_of\_web\_site])

- In the directory that was created above, included are: manage.py, db.sqlite3

- Start the mini Web-server: python manager.py runserver

- In any Browser enter this address: 127.0.0.0:8000

**- Creating first App:**

**---------------------**

- Create an App:

- python manage.py startapp [name of app]

(ex: python manage.py startapp music)

- New directory is created: music

- admin.py: add, delete user

- apps.py: configuration or setting file

- models.py: blueprint of database

- tests.py: Is for testing apps

- views.py: takes a user request and give back something

- App\_name/urls.py: is url mapping, which is writing as a regular expression

- On the same level as project create a new file, urls.py

- Import section : from . import views

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

r: regular expression

^: debut of string

$: end of string

views.[name] (ex. index)

name=[‘name’] (ex. index)

- On the level of the app:

- edit urls.py

- Import section add from django.conf.urls import include

- Add the url pattern :

(ex.: urlpatterns = [

url(r'^admin/', admin.site.urls), #default pattern

# If the user enters music then go to the project file:   
# project/urls file

url(r'^music/', include('music.urls')),

]

* App level (ex.music) edit views.py

- import section : from django.http import HttpResponse

from django.shortcuts import render

from .models import Album

- Create the the different methods for user requests

(ex. : def index(request):

# Conect to the database to recover all albums

all\_albums = Album.objects.all()

context = {'all\_albums': all\_albums,}

return render(request, 'music/index.html', context)

def detail(request, album\_id):

return HttpResponse("<h2>Details for Album Id : " + str(album\_id) + "</h2>")

**- Creating the DB:**

**------------------**

The database settings are in the setting.py file around line #77.

- To synchronized the code to the database : python manage.py migrate

- Create a models : models.py

class Album(models.Model): #Creates a table

artist = models.CharField(max\_length=250) # field name and type

album\_title = models.CharField(max\_length=500)

genre = models.CharField(max\_length=100)

album\_logo = models.CharField(max\_length=1000)

def \_\_str\_\_(self):

return self.album\_title + ' - ' + self.artist #print album title and artist

class Song(models.Model): #Second table

album = models.ForeignKey(Album, on\_delete=models.CASCADE) #links with the first table

**- Django API Shell:**

**-----------------**

- Note if you make any changes to the models; you will need to exit the shell and re-start python server, before any modifications will be active.

- To enter into the django Shell: python manage.py shell

To enter data into a table:

- Instantiate an instance of the class: a = Album()

- Use the dot method to assign value to fields : a.song\_title = “some song title”

- or, all fields at once: alb = Album(artist="Taylor Swift", album\_title="Red", genre="Country/Rock", album\_logo=<http://www.litlepups.net/f855d592ddde96d2.html>)

- When finish, save to db: a.save()

- Exit shell: exit()

- Make any and all changes to the models:

- Re-run the django API data shell : python manager.py shell

- from db\_name.models import table1, table2, …

- verify changes made to models

- Use filters : Table\_Name.objects.filter(field\_name) – Album.objects.filter(id=1)

- Filter with starts with: table\_name.objects.filter(field\_name\_\_startswith=’’ - Album.objects.filter(artist\_\_startswith='S')

**- Admin Interface:**

**------------------**

- create a super user :

- python manage.py createsuperuser

Username: admin

Email: [something@provider.com](mailto:something@provider.com)

Password: epsilon admin pw

- With a browser login Local server: 127.0.0.1:8000/admin

- If you want to see the tables in the DB, edit admin of the app:

- In import section: from .models import Album

- Register the models : admin.site.register(Album)

**- Connecting to the Database:**

**---------------------------**

* From the import section:

from .models import Album # From the views.py file import models Album

- define two methods:

def index(request):

all\_albums = Album.objects.all() # connecting to Database Album

html = ‘’ # variable for holding each link to each album

for album in all\_albums:

url = '/music/' + str(album.id) + '/'

html += '<H2><a href="' + url + '">' + album.album\_title + '</a></h2><br>'

url: concatenate /music/ + the string of album\_id

html += … : concatenate the html code with album\_id and field : album.album\_title

def detail(request, album\_id):

return HttpResponse("<h2>Details for Album Id : " + str(album\_id) + "</h2>")

**- Another View:**

**-------------**

A view is a function that return some html. Each urls pattern is connected to a view!

- adding another page to a website:

- Edit project\_name/app/urls.py (ex: Website\website\music\urls.py

- urlpatterns = [

# if only music is written on the website, the views.py is called

# with parameter index

url(r'^$', views.index, name='index'),

# /music/music\_id/ on the website, rp : album\_id is any number between [0-9] that can be repeated : + times

url(r'^(?P<album\_id>[0-9]+)/$', views.detail, name='detail'),

]

- Edit project\_name/app/views.py

- import

**- Template:**

**---------**

- project\_name/project\_name/app\_name/templates/app\_name/index.html

- from django.template import loader

- Create a Directory : from the App directory make s sub-directory : templates

- from Directory templates, make a sub-directory : music

- Create a file called: index.html(ths will be the home page of the app)

- From the views.py file, load the index.html: template = loader.get\_template(music/index.html)

- Create a variable: context = {‘all\_albums’: all\_albums, }

Return the template to the script : return HttpResponse(template.render(context, request)

- continuing with updating the web page :

- Detail page:

<img src="{{ album.album\_logo }}">

<h1>{{ album.album\_title }}</h1>

<h2>{{ album.artist }}</h2>

<ul>

{% for song in album.song\_set.all %}

<li>{{ song.song\_title }} - {{ song.file\_type }} </li>

{% endfor %}

</ul>

**- Raising a 404 HTTP Error:**

**-------------------------**

- project\_name/project\_name/app\_name/views.py

- from django.http import Http404

- project\_name/project\_name/app\_name/templates/app\_name/details.html

**- Adding to Database:**

**-------------------**

- To display the song title, Edit: project\_name/project\_name/app\_name/models.py:

-Under Class Song:

- Create a new method:

- def \_\_str\_\_(self):

Return(self.song\_title

- To make the change in the administrator panel:

- project\_name/project\_name/app\_name/admin.py

- Add Song to the import section (ex: from .models import Album, Song)

- edit the register section of file (ex: admin.site.register(song)

- To add records to a table there are two methods:

- Recover the id of a record(ex: alb\_1 = Album.object.get(id=x)

- Create an instance of the class(ex: song = Song())

- To list a records in a table – instance\_of\_obj.name\_of\_table\_set.all()(ex:abl\_1.song\_set.all())

- To enter data into table 1 :

- Recover id or primary key: var1 = Table\_name.object.get(id=rec\_num)

- Verify that the label is the one you wanted: var\_1.field\_name

- Create a instance of the object: (ex: song = Song())

- Tie the two table together using the foreign key: (ex:song.album = var\_1))

- enter data: db\_name.field\_name = ‘value’: (ex:song.song\_title = ‘Hello’))

- After all fields are filled in, save table:Table\_Name.save():(ex:song.save())

See Template Section of notes for the rest:

**- Hard coded URLs:**

**----------------**

- How to remove hard coded urls and why not to use them.

**- Namespace and HTTP 404 Shortcut:**

**--------------------------------**

- How to make sure that your app name space is unique: project\_name/project\_name/app\_name/urls.py:

- After the import section add app\_name = ‘name\_of\_app’

- in the template file : add ‘app\_name:view’(ex: in rhef, ’music:detail’

- 404 shortcut(views.py) :

- In the import section add: get\_object\_or\_404

- In method section: name\_of\_table = get\_object\_or\_404(name\_of\_table, id=primary key)

- See: Raising a 404 HTTP Error

**- Making templates Dynamic, by removing hard coding:**

**--------------------------------------------------**

- In the App/urls.py file, add the apps name after the import section of file: (ex:app\_name = ‘music’), also in the url section that where the name tag goes:(ex:name=’detail’)

- In a html file, add : {% url ‘music:detail’ album.id%}

**- Making Simple Form:**

**-------------------**

Background info for modifications: for this form; in front of each song, there will be a radio/checkbox button and at the bottom of page, a submit or validation button. When the user has selected a song to be one of their favorite and validates their chose by clicking the the submit button. The system recovers the song\_id & the album\_id and verifies if album & song exist, if not, send a 404 message to user. Else the system places a star logo at the end of the song and update the DB for that song.

- Edit the models.py file and will add a new an attribute to the song method.(ex:is\_favorite = models.BooleanField(default=False)

The models.py file has been modified; need to execute the migration again. Note models.py is the structure of the database!

- python manage.py makemigrations name\_of\_app ; makes the change(s) to the blueprint of the db

- python manage.py migrate : This applies the change(s) to the db

- Create a urls or pattern map: (file: music/urls.py)

- Basic url patterns :(ex:

urlpatterns = [url(r'^P(<album\_id>[0-9]+)/favorite/$', views.favorite, name=' favorite '),

]

r: regular expression

^: debut of string

?P:group name

(<field\_name>

[0-9]: any digit between 0 – 9

+: repeat

$: end of string

views.[name] (ex. index)

name=[‘name’] (ex. index, is the shortcut name)

**- Including CSS, Bootstrap in Static files:**

**-----------------------------------------**

When you would like to include logos, images and special characters you use static files

- Create a sub-directory of project app:(project\_app\static)

- Create sub-directory of static :(ex:static/name\_of\_app/project)

- To organize your files you can create another sub-directory, for your css, images, etc. files

- For CSS files, create a stylesheet file: style.css

This style sheet will include any valid CSS code!!

- Add an image to background of the app, download an image and copy it to image directory.

- Edit the style.css file and add a background to the app:

- body { background: white url(“images/name\_of\_background”) }

- Include the CSS file to code:

- Edit index, need to load path of static files to index.html template: use the special template syntax: {% load staticfiles %}; Django needs to know where the exact location of the file is, use the html code of a link: <link rel=”stlyesheet” type=”text/css” href=”{% static ‘music/style.css’ %}” />

- If you would like add bootstrap framework to your code, here is the link: <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

- Add a menu bar using bootstrap:

# navbar-inserse or navbar-default

<nav class=”navbar navbar-inserse”>

<div class=”container-fluid”>

<div class=”navbar-header”>

<a class=”navbar-brand” href=”{% url ‘music:index’ %}”>Viberr</a>

</div>

</div>

</nav>