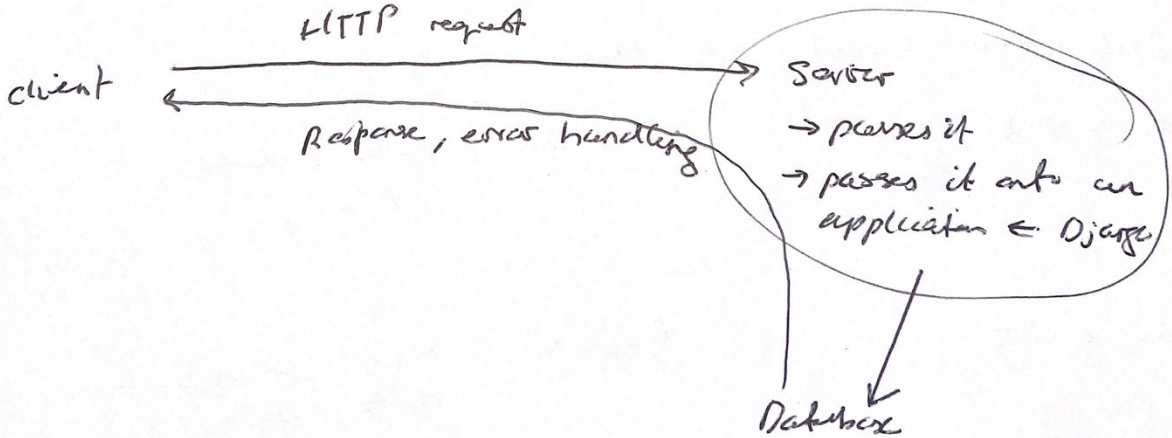
Django Using Generative AI Help

★ COURSE OVERVIEW

→ generative AI help
→ open source web framework
→ ORM ← Object Relational Mapping system
→ for web applications with Python
→ MTV ← model template view architecture
  - model ↑ data
  - template ↑ layer
  - view ↖ integration
→ chat GPT can be used to help install it
→ you can make web apps using it
→ static assets ← HTML pages
→ prompt engineering help w/ it

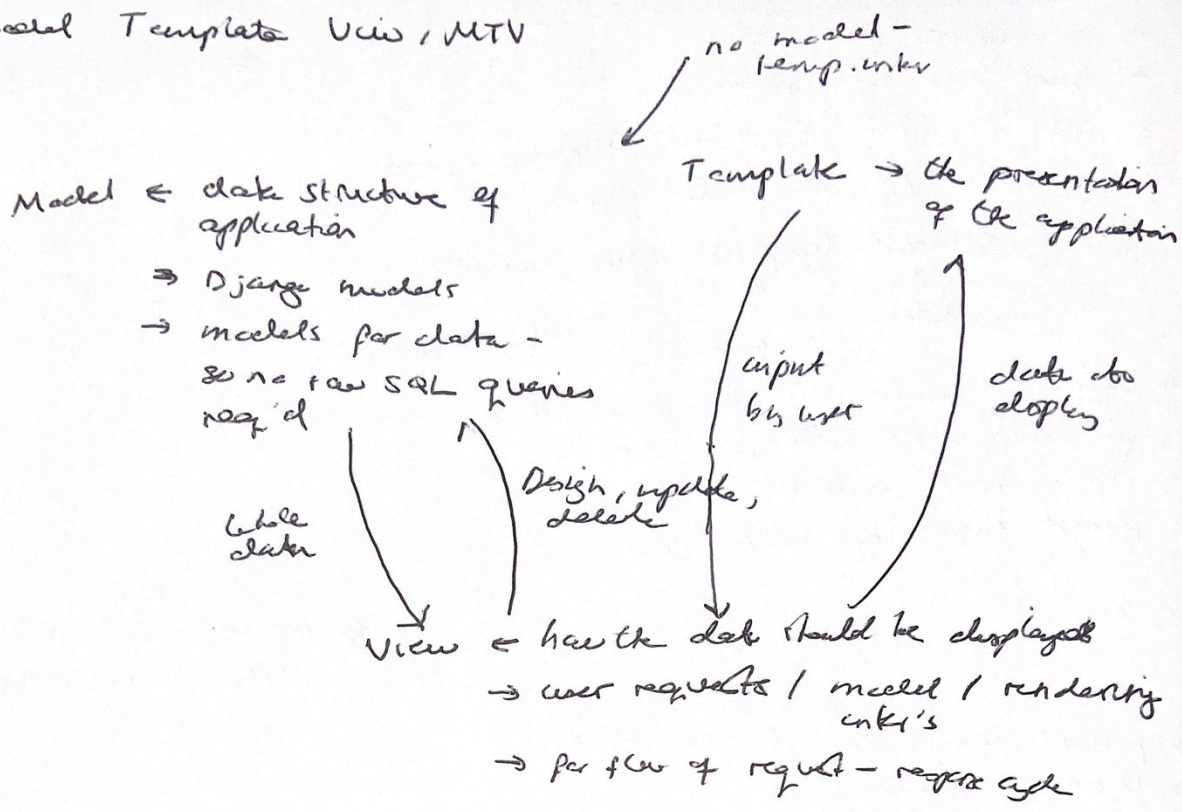
★ DJANGO FOR WEB APPLICATION DEVELOPMENT

→ prompt engineering w/ gen. AI tools
→ Django "framework"
  → Python web framework ← for websites w/ Python
  → has libraries
  → MVC ← model view controller (std.)
  → data, components
→ ~~HTTP~~ request response control flow



→ Django is server-side, Python for the web applications

→ Model Template View, MTV



→ DJANGO RUNS ON THE SERVER, USES MVC ARCHITECTURE = MODEL VIEW CONTROLLER

→ for views - templates

spec. ones — static assets (html + js)

→ sending response obj's back to client

→ Model - View - Controller is Model - Template View

model - data, business logic	model - data + database op's
view ← GUI	template - defines presentation layer, data display

- ORM = object - relational mapping
- Django has URL routing
- it has a template engine, authentication system.
- security feature is SQL injection

- a testing suite

☆ PROMPT ENGINEERING TO LEARN DJANGO

→ web applications + for prompt engineering
↑
input instructions w/ NLP
→ a prompt ← after info is entered (into a prompt)
→ engineering them
→ generative AI ← generative models can create different formats of output

→ you can use ChatGPT / Google Bard / Microsoft Bing
→ prompting ← so it knows what output to give
→ what kinds of outputs we want - bias in outputs
→ hallucination minimized by guiding the model
→ safety / prompt hacking

#1 Instruction
#2 Context
#3 Input data ← nds / wds.
#4 Output syntax

← prompt engineering features
- white code
  - gen. / debug / explain code
  - to ↑ code performance
  - to translate bet coding lan's

- outline
  - writing code w/ prompts
  - debugging / explain prompts
  - ChatGPT / Bard / Bing chat ← can make outlines

☆ USING GENERATIVE AI ASSISTANTS

- w/ Django
- ChatGPT to do it
- ChatGPT / Bard / Bing ← ChatGPT best for Django.
- GPT 4 / 3.5 ← ↑ lan. models, 4 = ↑ accurate
- there are different plans for it → pric. access to GPT's servers
- Google Bard - free, gmail req'd
- Bing chat ← GPT's open AI model used here
- hallucinations = model makes something up

- in ChatGPT
  - Django = Python framework. for web applications
  - Flask = o.e., CherryPy
    ↑
    ~~test~~ of use
    case
  - Django = ↑ powerful
  - Django = for ↑ er, flask = for ↓ er web application
    ↑ most pop. one ↑ neck of a bottle (narrower), HTML/JS
  - IDEs ← Sublime / VS code - asks chatGPT for it
- HTML / CSS / SQL
- Django ← guitarist named after

☆ PAIR PROGRAMMING TO CREATE A DJANGO PROJECT

- making a website w/ Django.
- uses ChatGPT instr's, Sublime vs VScode
- virtual env. is a sandbox, ~ a VM in case of mistakes
  → pip install Django.
  → then creating the proj
- in terminal
  python — v — version
  mkdir for-project ← makes proj. dir, cd's → it
  → cds in there

python3 -m venv django_env ← makes Django. VM. in folder we cd'd →

- she is on Windows
- uses GPT comm to activate the env.
  → pip install Django ← to install the Django web f.wk.
  → she goes thr GPT instr's
  → django-admin --help
  ↑ to enter w/ proj.
  → migrations
- then creates django proj. ← catalog-mgnt
  → entire web application
  → multiple apps working togeth
  → catalogue man. ← this is a folder, w/ nested subfolders (segment)

- the proj needs to be formatted correctly
  → it's a Djang proj
  → GPT explains all files in proj.

  - - init - - .py ← empty py file
  → treat it as a package

  asgi.py ← asynchronous server gateway interface - deploying Django w/ asgi
  wsgi.py
  ↑
  web server gateway interface ← managing wsgi deployment

  settings.py ← for settings for proj., inc's base dir
  → config settings for installing apps
  → Djangp app components
  → templates for html, css
  → the template configs, database configs (sqlite)
  → location of static files

  urls.py ← url location mapping to view posts
  → the paths to the administrate panel

  manage.py ← marked as cut
  → a command line utility script (for terminal)

  django-admin ← across proj's

★ RUNNING THE DJANGO DEVELOPMENT SERVER

- we have a Django proj + structure
- cd's into it
- catalogue main folder
- manage.py ← for performing tasks
  → running the server on the local mach. in console
  → can get migration / admin errors
  - settings.py file ← apps provided by default
  - an administrative panel + default apps
    admin, auth, contenttypes, sessions
  - to adapt the database str.

- it returns a url for the server in the terminal
  - there is a default site for this then
    .../admin ← url ⇒ it asks for user/passwd admin
    → only for admin (not categories/products etc.)
    → in urls.py, tells us which

- to run it on a different port ← she asks ChatGPT
  - changing the port ~~on the~~ the server is running on
  - now it's said in the URL
  - she's restarted it in the terminal

- catalog_mgmt > db.sqlite3 ← for sql database of proj info
  → empty @ 1st :: no migrations
  → sqlite = no server ⇒ ↙ ⇒ med. applications (can Δ)

☆ CREATING A DJANGO APP

→ the Django dev't server is default here
  → the URL for the port is for a generic site
  ⇒ no app
    → apps have their own subfolders
    → catalog_mgmt ← admin for whole proj

- a view in Django:
  - HTML, Python
    ↑ view func.
  - returning an HTTP response
    else ...
    response
  ] ← AE can give you incomplete cos's you think are incomplete
    → unless exp.

- she creates a views.py file
  → ↙ catalog_mgmt
  → defs a func. → home(request)
    ↑ except. for ↘ HTTP request
    return HttpResponse("...")
  → the func. returns an HttpResponse = a Python func.

→ then in urls.py
→ adds path('', views.home, name='home') ← name for the URL path (optional)
↑ gets. ↑ py func. to run view @ that URL

→ Refreshes the page
→ in urls.py, imports from views file (.py)
→ creates comp's for the app
→ apps in the proj

→ sets up an app w/ GPT
→ in bash
→ init's the app as a part of the installation
→ catalog_mgmt ← creates proj config files
→ follows GPT's instr's for new app
- makes subfolders
- contains ↑ Python subfolders

→ under migrations ← now we have a new __init__.py file
↑ treat it as a package

→ admin.py ← to customize the model
→ apps.py ← config file for the app
→ models.py ← for database entities
→ orm ← objects relational mangement
→ views.py ← for view func's / classes

→ it's in app name & settings.py file

☆ ~~creating and~~ CREATING AND CONFIGURING VIEWS IN A DJANGO APP

→ we have a Django app and are setting up views
→ She asks ChatGPT (Bard)
→ create a views func. in view.py
→ then added a url pattern in urls.py
→ adding it to urls.py and importing it into this
→ following Bard's instructions
→ in views.py, she def's a func.

→ def app.home (urls - request)
→ She adds the url path for this to urls.py (under urlpatterns
→ then gives it a name

- in the catalogues mgmt folder
  → She goes to the ~~test~~ url path for the app
  → it /catalog/ home returns a 404 not found error
    - management urls
    - the path with name = home ← it looks into the urls folder for the path, but not the app

→ url patterns in the projects path
- there is a url.py file for the entire project, and the app

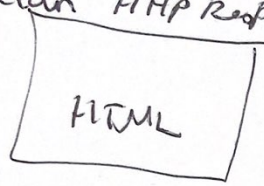
  → url patterns
  → path ('catalog/', include('catalog.urls'))
  path for url pattern ← include url configs from other catalogues / configurations
  → the url pattern is a path on the server
  → each app has its own set of urls
  {home - url} / catalog / home

- She goes to the url for the app and it's working
  - there is also another case for 404 errors ... / catalog /

→ then she sets up several views
  → app home / product / about
  → each is a func. in views.py and returns an HTTP response
  → then she adds them to the path of urls.py, ✓ url-patterns
  → same syntax as
  → these urls can be tested in the browser

☆ USING TEMPLATES & STATIC FILES WITH GENERATIVE AI HELP

→ views are html pages
→ they display html
→ from django.shortcuts import render
→ under views.py func's, we puts html in the return obj's
  ↳ views.py
  → def ... ( ):
    return HttpResponse (



)

→ HOW HTML IS SERVED VIA A .PY FILE
→ as a return HttpResponse obj in a .py file

- this is not as scalable ← css
- goes to chatGPT for help, pastes it in
  → Django templates files, render func.
  → then modifying the views.py file
  → the url func.
  → to put the home.html file ← spec. dir structure
  → creates a templates sub folder ⇒ (temps dir.
    ⇒ then a sub folder ↙ templates, spec. for this app
    → its still ↙ catalog - mgmt
    ⇒ then she deploys home.html ↙ that ~~the~~ templates dir.
    → she deploys another, for products.html, + about.html

→ then in views.py, we have def .... ( ):
Ex return render(... , ...)

→ rendering html templates we're rendered
→ this is relative to the templates folder in the app
→ then she goes to the url in the browser, which for the different html temp files
→ all the html files are in catalog
~~→ to get in css~~

→ to get in css
→ she asks ChatGPT (Bard) for css in Django
→ css in a Python framework
→ we have a reference for the static dir.
→ instr's for css styles
{% %} ← template tags - flow of logic

→ sets up a new statics folder
→ static folder, catalog subfolder
→ all css files in some folder

Static > catalog > css

→ then styles.css in there
→ updating template files
→ she links the css file to the html and we already have
→ then can be seen by refreshing the webpage
→ if it works, the styles r already been
~~cached~~ cached

☆ DEALING WITH MISDIRECTION FROM GENERATING BAD TOOLS

→ we have a web application w/ html template files
○ static assets in the static folder
→ the css styles file is linked into an html file e.g Lec
→ Bard suggests another approach
  → template tags / refactoring code
  → making ∆'s in one place, e.g
  → ↑ portable code
  → without ∆ing the template code
  ⇐
→→ then in the settings.py file
  → all the static assets are listed
  → template tags
    <link rel=... .css> ← ∆'ing this for a template tag
  → referencing the style sheet
  → there is a load static template tag in the file
    href=" {% static '....css' %}
    → template tags in the <link> tag in html which links it to the styles. css file

→ debugging w/ Bard
  → you can put error messages → Bard
  → static isn't recognised by Django
  → the static files tag
  → alt., Bing chat
    {% load staticfiles %} here
    → Bard returned a 4 result
    → the uses multiple APIS to check code

→ load static

→ load static is a future-proof approach
→ Static vs Static files
→ referencing static assets
→ chatbots can be wrong ← generative AI's
→ are always confident
→ check what they give you

→ the 5's all html pages
- it's a ℰ to load Static e.g. line @ top of each
- they are loaded in an app that comes loaded in Django.
→ then refreshing the url to see if this adds

→ displaying static assets, e.g. img's
→ Bard gives the html, which she pastes in
→ there is an src attribute in an img > html tag
→ the 5's an image in
→ the image needs to exist @ the path on the server, or it won't show on the loaded page
→ images is a sub-folder on the same root level as the CSS folder

→ she inspects the html this produces and it can be seen there

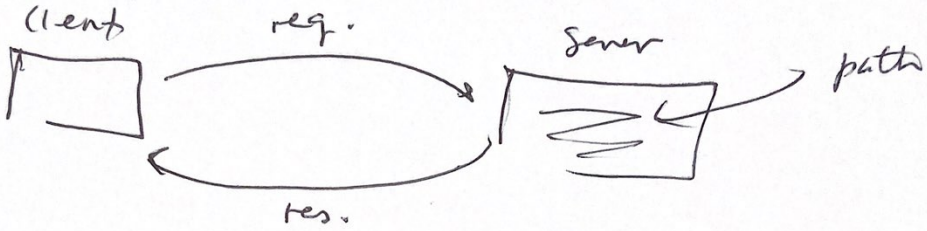
☆ Capet summary

→ Django ← web application dev't
→ model ← data & schema
→ Template ← view (presentation layer, html files)
→ view ← user requests/responses
→ then a Django app + generative AI to check it
→ styling the proj. 5's.
→ built-in styl. libs.
→ views / templates in Django.
→ Static files, template tags

→ Django basics, views / prompt engineering (GPT)

★ QUIZ

1. HTTPResponse("....")



3. python manage.py runserver 8080 ← to run port 8080

5. Model, View, Schema