# **Django Market Tutorial**

**Continue from 1:27:00 – Displaying Item Objects in Items.html**

[Python Django Crash Course for Beginners | Create a Marketing Website in 3 hours - YouTube](https://www.youtube.com/watch?v=qMrAFscMBBc&ab_channel=JimShapedCoding)

# **Backend Engineering:**

Django is a backend tool allowing the routing of views via the urls.py and management of database. It does not act as the front end, but simply facilitates the management of the front end via HTML, Javascript etc.

It can be combined with other database systems such as SQL and front end frameworks such as Vue.

# **Commands:**

## **Django-admin startapp main = Creates a new Django App**

## **Python manage.py migrate – instantiates the database tables**

# **Template Tips**

## Importing html files to the base template – Include function {% include %}

Instead of copy pasting lots of html into the base template. This is not the same as extending a template with a codeblock.

Create new folder **includes** in the **templates/main folder.** Which will hold all the small html files for the base template.

A screenshot of a computer

Description automatically generated with medium confidence

  <body>

      {% include 'main/includes/navbar.html'%}

# **Django Lifecycle Management**

## **Lifecycle 3: With embedded html string**

How to route view URLs - The process how views are routed to via the URLs.py file

**A picture containing diagram

Description automatically generated**

## **Lifecycle 2: With html templates**

We create a view, rendering an html template that we are pointing towards in the view template index.

The view is being pointed towards from the urls routing, based on the url that the client is hitting i.e. /home.

We need an html front end to display the content in the html template, which is being redirected from the views.py file. A picture containing text

Description automatically generated

## **Default .py files:**

### **Settings.py**

Configuration file to deal with requests and security

Debug = True -> shows us all the error messages in detail

### **Urls.py**

Are the routes that are configured on your website

i.e. create a route that goes to \home

A screenshot of a computer

Description automatically generated with medium confidence

## **Main Application .py files:**

## **Init.py**

Convention file that specifies that the whole main directory is a package that the Django project will use. Don’t touch this.

## **Admin.py**

Responsible to fill in information to the auto generated administrator page. Customise the information within your website.

## **Apps.py**

Configuration files that specifies this main directory is an application within the project.

## **Views.py**

Responsible for dealing with response request architecture. Write python functions that will get content that we want to display on our pages.

i.e. facebook, you as a client requests functions on a web server. Those functions we are going to create.

## **7. urls.py (created by us)**

Create this urls.py to be organised when creating the routes. It is recommended to create urls.py for each app in Django.

## **Application vs Project**

Application is a divider between all the components that you want to write on your website.

I.e. Instagram has stories and posts. These components are seperated into their own “applications”

# **Chapter 3 Create Your First Page**

## **Route Main URLs in Market**

We need to make the market app aware of the urls from the Main app, which is done in the urls.py

Before

A screenshot of a computer

Description automatically generated with medium confidence

After

Text

Description automatically generated

## **Creating the first path or route**

We write path functions in the list, which are paths that are accessible and will visualise a page.

We will route to a specific page, and point a specific view file to the routing url.

#### Remember: Install the main application in settings.py

'main.apps.MainConfig',

Text

Description automatically generated

# **Chapter 3 - Django Templates**

How to link htm files to the view file, instead of embedding the html directly in the view file. The benefit of this is that: we don’t want to cinlude all the html in a single string.

**Text

Description automatically generated with medium confidence**

## **Rendering htmls in views.py**

Instead we want to create a function that will direct the view into a html file.

The render function exists to render the view into a html template.

***Rendering Function:***

***Request:***

We will insert a request argument which will pass all the requests from the client and pass it to the HTML template.

***Template\_name:***

Template stands for all the html files that we are keeping in a separate directory to keep everything organised.

Text

Description automatically generated

## **Templates Folder**

Views are by default are referred as a directory called template, which is a convention that Django understands where to render its templates.

There is a convention that forces us to create a new directory which will be named templates.

***Templates (folder in the main directory)***

***Main (folder in the templates directory)***

**Text

Description automatically generated**

# **Item Page**

**Displaying item objects in the items.html page**

**Create a dictionary in the views.py which will hold all the “item” objects and we can call it from the html template file.**

**A screenshot of a computer

Description automatically generated with medium confidence**

## **Passing information to the html tempalte from the views file**

**I want to work with this dictionary in the html template, and make sure to include it with a variable name “items”, using the context argument.**

**You name the key of the first item in context, the name you want to use in the html template. Always ensure the key the same as the value here, referring to the items dictionary.**

**A screenshot of a computer

Description automatically generated with medium confidence**

## **Using the dictionary in the html template**

**Special template language called ninja template, that can be used in the html template to access the dictionary items.**

**Text

Description automatically generated**

## **Outputting the dictionary data without styling**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Graphical user interface, text, application

Description automatically generated**

## **Outputting the dictionary data using html functions**

Using a for loop in html we can iterate through each item and display them in a style

Text

Description automatically generated

**Graphical user interface, text, application

Description automatically generated**

# **Template Inheritence**

Avoiding

Create a base template in Django, and reuse it in all of the html templates. Create a new html template and import everything to the other html templates based on the base tempalte.

Such as for navigation bar, we want to make sure it is included in each of the pages, but we don’t want to repeat the html code for each navigation page.

## **Creating a base template file**

Text

Description automatically generated

Text

Description automatically generated

## **Using the base template**

Extending the base template, so that we can use it in the home.html template

Graphical user interface, text

Description automatically generated

## **Reusing the base.html template**

In order to reuse the base tempalte we create a code block in the base template, which is used for a specific html template. We can also name the code blocks, such as content.

The block gives a special area, which is dynamic, that can be called in a specific html tempalte. You can write unique html code within your template.

This is how you organise your templates, within Django. You grab a base template, and extend it within your specific html template, and create a block which will be unique for the specific html.

# **Building Templates**

# **Page Header Template (Explain what website is about)**

[What is the Bootstrap page header ? - GeeksforGeeks](https://www.geeksforgeeks.org/what-is-the-bootstrap-page-header/#:~:text=What%20is%20the%20%EE%80%80Bootstrap%20page%20header%EE%80%81%20%3F%20%EE%80%80Page%EE%80%81,is%20always%20the%20first%20impression%20of%20the%20viewer.)

[Bootstrap Headers - examples & tutorial (mdbootstrap.com)](https://mdbootstrap.com/docs/standard/navigation/headers/)

As header is always the first impression of the viewer

So, one would want that their page has a good-looking header at the top of the page and the header can be highlighted easily.

Page header is used to add suitable spacing around the headings on a page and adds a horizontal line under the heading.

{% extends 'main/base.html' %}

{% block title %}

    Ahmed's Market- HomePage

{% endblock %}

{% block pageheader%}

<div class="position-relative overflow-hidden p-3 p-md-5 m-md-3 text-center bg-light">

    <div class="col-md-5 p-lg-5 mx-auto my-5">

        <h1 class="display-4 font-weight-normal">Ahmed's Market</h1>

        <p class="lead font-weight-normal">Start purchasing products by clicking the link below</p>

        <a class="btn btn-outline-secondary" href="">Get Started</a>

    </div>

    <div class= "product-device box-shadow d-none d-md-block"</div>

    <div class= "product-device product-device-2 box-shadow d-none d-md-block"</div>

</div>

{% endblock %}

# **Linking <a> tag to template (items)**

**Don’t hardcode the link:**

You don’t want to hardcode the routing link, because you would need to go directly into the template and change it each time. Like below we have it routed to /items. Not recommended.

<a class="btn btn-outline-secondary" href="/items">Get Started</a>

Tip: You don’t want to hardcode the routing link, because you would need to go directly into the template and change it each time. Like below we have it routed to /items. Not recommended.

**How to do it: Codeblocks**

**Use code block and pass in the name of the url from urls.py into the codeblock as a string ‘’**

 <a class="btn btn-outline-secondary" href="{% url 'items' %}">Get Started</a>

Use a codeblock linking to the url name “items”

from django.urls import path

from main import views

urlpatterns = [

    path('home/', views.homepage, name='home'),

    path('items/', views.itemspage, name='items'),

]

# **Build Navigation Bar**

**Text

Description automatically generated**

     {% include 'main/includes/navbar.html'%}

      {% block pageheader %}

<nav class="navbar navbar-expand-md navbar-dark bg-dark">

    <a class="navbar-brand" href="{% url 'home' %}">Ahmed's Market</a>

    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

        <span class="navbar-toggler-icon"></span>

    </button>

        <div ="collapse navbar-collapse" id="navbarNav">

            <ul class="navbar-nav me-auto"

                <li class="nav-item">

                    <a class="nav-link active" href="{% url 'home' %}">Home <span class="visually-hidden">(current)</span></a>

                </li>

                <li class="nav-item">

                    <a class="nav-link active" href="{% url 'items' %}">Items</a>

                </li>

            </ul>

            <ul class="navbar-nav">

            </ul>

        </div>

</nav>

# **Django Models: Build Database With SQLite**

It is not maintainable to hardcode the data into Django files such as the views.

**Hardcoded Data in The View:**

def itemspage(request):

    items = [

        {

            'name': 'Phone',

            'price': '500'

        },

        {

            'name': 'Laptop',

            'price': '1000',

        }

    ]

## **Object Relational Mapping (ORM)**

Translating database model to a Django model, is called Object Relational Mapping. It allows us to programmatically create tables for the database.

Text

Description automatically generated

## **Fields – Consider Using Image Field**

[How To Upload Images With Django - Django Central](https://djangocentral.com/uploading-images-with-django/#:~:text=The%20image%20column%20is%20an%20ImageField%20field%20that,dynamic%20paths%20for%20the%20pictures%20is%20also%20possible.)

## **How to Instantiate the Database – python manage.py migrate**

## **Step 1: Run python manage.py migrate**

Will instantiate the tables

## **Step 2: Run command python manage.py makemigrations**

Will store the actual database

PS C:\Dev\Django\Tut\market> python manage.py makemigrations

Migrations for 'main':

main\migrations\0001\_initial.py

- Create model Item

## **Step 3: Run command python manage.py migrate**

Make the database tables of the migrations

PS C:\Dev\Django\Tut\market> python manage.py migrate

Operations to perform:

Apply all migrations: admin, auth, contenttypes, main, sessions

Running migrations:

Applying main.0001\_initial... OK

# **Django - Admin Page (go to /admin)**

## **Run command: python manage.py runserver**

Graphical user interface, website

Description automatically generated

Graphical user interface, website

Description automatically generated

## **Create Super User: (User with all permissions to customise data)**

## **Run command: python manage.py createsuperuser**

## **Register Data Models to Administrator page (admin.py)**

Text

Description automatically generated

from django.contrib import admin

from main.models import \*

# Register your models here.

admin.site.register(Item)

Graphical user interface, application, website

Description automatically generated

## **Customising the data name object in Admin page – from the modesl.py**

Customises the way the object is shown to you, which is modified in the models.py

from django.db import models

# Create your models here.

class Item(models.Model):

    name = models.CharField(max\_length=255)

    price = models.IntegerField()

    description = models.CharField(max\_length=1024)

    image\_url = models.CharField(max\_length=512)

    def \_\_str\_\_(self):

        return self.name

# **Displaying the table data in a template (from views.py)**

Render the objects from the items model

We need to import the models in the views.py file

We write a function that stores the Item table objects into a items variable (object).

We render the item objects by writing a context function, and storing the item variable into a dictionary, with a key value pair, ‘items’ : items

from django.shortcuts import render, HttpResponse

from main.models import Item

# Create your views here.

def homepage(request):

    return render(request, template\_name='main/home.html')

def itemspage(request):

    items = Item.objects.all()

    return render(request, template\_name='main/items.html', context={'items' : items})

# **Displaying Item Objects in items.html (iterating through items)**

WE use a table element in html

## **Using the Table Tag <table> in html**

[Tables · Bootstrap v5.1 (getbootstrap.com)](https://getbootstrap.com/docs/5.1/content/tables/)

TABLE TAG

Table of multi-dimensional data. The HTML Table Element (<table>)

represents tabular data: information expressed via two dimensions

or more.

Attributes (modiတဠers)

summary + global attributes

## **Iterating through Database Model and displaying them in the template**

## **Displaying Images from Url in table**

                <td>

                    <img alt="image" style="width:64px" src="{{item.image\_url}}">

                </td>

                <td>{{item.price}}$</td>