The background features abstract, overlapping geometric shapes in various shades of purple and blue, creating a modern, layered effect. The shapes are primarily triangular and polygonal, with some areas appearing more translucent than others.

Course Module 1: Course Unit 11: Django (Advance Concepts)

Week 14

The background features abstract, overlapping geometric shapes in various shades of purple and blue, primarily concentrated on the right side of the image. The shapes include triangles and polygons of different sizes and orientations, creating a dynamic, layered effect. The colors range from light lavender to deep indigo and dark blue.

Django

Django

- high-level Python web application framework (enables the rapid development of web applications)
- backend framework used to resolve problems of connectivity with databases, other server problems, SEO (Search Engine Optimization) solutions



Django

Intermediate Level

Topics

Django Intermediate Level Topics

- **Django Admin Interface**
- Django Database
- Django Redirect
- Django Cookies Handling
- Django Form Handling and Validation
- Django File Upload
- Django Static File Handling
- Django Bootstrap
- Django CRUD

Django Admin Interface

Admin interface

- the control panel of your website/ project
- used by the administrators of the website to add, delete and edit the content on the site
- maintain other functionalities accordingly made by the developers

***Admin interface being the main controller,
it is needed to be coded by the developer
individually.***

Django Admin Interface

Django Admin

- the preloaded interface made to fulfill all the needs of the developers.
- Its language is quite generalized rather than technical.
- *a fully featured interface and you won't need to write an Admin interface for your project*

Steps To Use The Django Admin Site And Implement User Models: [Django Admin Interface - Setting Up Django Admin Site in 12 Easy Steps - DataFlair \(dataflair.training\)](#)

Django Intermediate Level Topics

- Django Admin Interface
- **Django Database**
- Django Redirect
- Django Cookies Handling
- Django Form Handling and Validation
- Django File Upload
- Django Static File Handling
- Django Bootstrap
- Django CRUD

Django Database

*Whenever we are creating a web project or any kind of project, we want some kind of input by our end-users or consumers. All that data/ input is handled by a **Database**.*

Django Database

Connecting Databases with Django Project

By default, when we made our first app and started the server you must have seen a new file in your project directory, named as '**db.sqlite3**'.

- The file is database file where all the data that you will be generating will be stored.
- It is a local file as Django is a server-side framework and it treats your computer as the host when you actually run the server in command line/terminal.
- This file is generated automatically because Django has a default setting of the database set to the SQLite.



myblog



write



db.sqlite3



manage

django Database

1

**DATABASE
Dictionary
Indexes**

2

**Connecting
MySQL database
with Django
Project**

Django Database

1. DATABASES Dictionary Indexes

Firstly, open the settings.py file of your web-application/ project and there find this part.

```
# Database
# https://docs.djangoproject.com/en/2.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Django Database

1. DATABASES Dictionary Indexes

This partition has information regarding the connection to the database.

DATABASES is a pre-defined dictionary in Django Framework with the 'default' as an index having the value for the main database where all the data is to be stored.

```
# Database
# https://docs.djangoproject.com/en/2.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```


Django Database

1. DATABASES Dictionary Indexes

The default is holding a dictionary where there are 2 indexes:

- ENGINE
- NAME

```
# Database
# https://docs.djangoproject.com/en/2.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Django Database

1. DATABASES Dictionary Indexes

- ENGINE - specifies the library to be used when connected to a certain website. In the value, we have to put the file, “**django.db.backends.sqlite3**”, which is the python library for sqlite3 database and will translate your python code to the database language.
- NAME - **name** of the database that you are using and the **location** of your database. This parameter changes according to the type of database you are using. Passing the name of the database file or if the file is not present, this will create the db.sqlite3 file. If you change the name to db1.sqlite3 or anything of your choice it will create that file in your root directory every time you run server again.

Django Database

1. DATABASES Dictionary Indexes

Every database has some attributes which actually become the default dictionaries indexes which you can change/ create according to the database you are connecting to.

Django Database

2. MySQL and Django – Connecting MySQL Database with Django Project

MySQL is a very powerful database providing you with tons of features and flexibility.

Steps to integrate Django project with MySQL:

- i. Install Xampp (free opensource tool which provides you with the Apache server and phpMyAdmin which is the best source for beginner programmers to work with MySQL.)
- ii. Run Xampp Control Panel
- iii. Creating a SQL database
- iv. Modifying settings.py

[Django Database - How to Connect MySQL Database with Django Project - DataFlair \(data-flair.training\)](#)

Django Intermediate Level Topics

- Django Admin Interface
- Django Database
- **Django Redirect**
- Django Cookies Handling
- Django Form Handling and Validation
- Django File Upload
- Django Static File Handling
- Django Bootstrap
- Django CRUD

Django Redirect

*Websites, in general, have **URL redirection** to enhance the user experience and it's necessary functionality to have in your websites. URL Redirection helps you gain control of your website's flow; Django allows you to easily redirect URLs.*

Django Redirect

Need for URL Redirection

- When you try to access a social media page where you want to message or like the post, then the server redirects you to the login page if you haven't done that before.
- Not only that, some good developers have made the system that, when you sign-in/sign-up, you reach to the page you wanted to access before.
- When you are performing some important operation on the website, like changing your password. The website should redirect you to a page showing you that the operation was successful.

Page Redirection is an important factor in providing a great user experience to the user. A developer will make good use of this feature of Django.

Django Redirect

The `redirect()`

This function is capable of taking more than just URLs, you can redirect your users to models, redirect to a particular URL, while also passing the key values and other needed information.

When our view function is only redirecting to a certain URL, without performing any function then, Django has provided us with a special Class:

```
django.views.generic.base.RedirectView
```

Django Redirect

Attributes of RedirectView:

.url – If this attribute is used, then the value should be URL string which can have placeholders from python and they can be changed dynamically.

.pattern_name – This is the collection of URL patterns, its like urlpatterns list for redirects.

.permanent – This takes Boolean values of true and false. If it's true, then the redirect becomes permanent. By default its value is false.

Django Redirect

Attributes of RedirectView:

get_redirect_url(arguments, keyword_arguments)

– This is a method that returns a string in the form of URL. this method is responsible for generating your URL, Django allows the developer to overwrite this method any way they want.

.query_string – This attribute takes values, true or false and by default it's false. If this attribute is set to true then the view will add or append any query_string to the URL and return the same.

Django Redirect

Advanced Usage of Django Redirect

redirect() can handle most of our simple redirection process which is perfect, but there are some use cases where the simple **redirect()** is not enough.

There are some more ways for using the **redirect()** for advanced control over your website:

1. Passing any parameter with **Redirect()**
2. Special Redirect Codes

[Django Redirects - The Essential Guide of URL Redirects in Django - DataFlair \(data-flair.training\)](#)

Django Redirect

Limitations of Django Redirect

1. Redirects that doesn't have any effect

`django.shortcuts.redirect()` is very easy to use. `redirect()` just returns a response object. This response object shall be returned from our views component otherwise the redirect won't work.

2. Infinite redirect Cycle

The infinite redirect cycle is when two redirects are pointing towards each other and the server just ends up redirecting over and over without stopping.

[Django Redirects - The Essential Guide of URL Redirects in Django - DataFlair \(data-flair.training\)](https://data-flair.training/tutorials/2020/05/django-redirects.html)

Django Intermediate Level Topics

- Django Admin Interface
- Django Database
- Django Redirect
- **Django Cookies Handling**
- Django Form Handling and Validation
- Django File Upload
- Django Static File Handling
- Django Bootstrap
- Django CRUD

Django Cookies Handling

Cookies, technically called HTTP Cookies are small text files which are created and maintained by your browser on the particular request of Web-Server.

They are stored locally by your browser, and most browser will also show you the cookies generated in the Privacy and Security settings of the browser.

Django Cookies Handling

HTTP is a stateless protocol. When any request is sent to the server, over this protocol, the server cannot distinguish whether the user is new or has visited the site previously.

Suppose, you are logging in any website, that website will respond the browser with some cookies which will have some unique identification of user generated by the server and some more details according to the context of the website.

Cookies made these implementations possible with ease which were previously not possible over HTTP implementation.

Django Cookies Handling

Cookies work like other HTTP requests over the Internet. In a typical web-system, the browser makes a request to the server. The server then sends the response along with some cookies, which may contain some login information or some other data.

When the browser makes a new request, the cookie generated previously is also transmitted to the server. This process is repeated every time a new request is made by the browser.

The browser repeats the process until the cookie expires or the session is closed and the cookie is deleted by the browser itself.

Django Cookies Handling

The cookie applies in all sorts of tasks, like when your login to a website or when shopping online on the web.

Google AdSense and **Google Analytics** can also track you using the cookies they generate. Different websites use cookies differently according to their needs.

Django Cookies Handling

Creating Cookies in Django

Django bypasses lots of work which otherwise would be required when working on cookies. Django has methods like **set_cookie()** which we can use to create cookies very easily.

The `set_cookie()` has these attributes:

- **name:** It specifies the name of cookie.
- **value:** It specifies the text or variable you want to store in the cookie.
- **max_age:** It is the time period of cookie in seconds. After the time period completes, it will expire. It is an optional parameter; if not present then the cookie will exist till the time browser close.

Add this method to our view functions for creating cookies.

[Django Cookies and Cookies Handling - How to Create Cookies in Django - DataFlair \(data-flair.training\)](#)

Django Cookies Handling

Read Cookies from request

Cookies pass to the particular website every time the client makes a request. Therefore, server every time receives a cookie alongside the request. Django makes it very easy for us to retrieve data from a cookie.

1. Using request.COOKIE[]

Syntax: request.COOKIE['cookie_name']

COOKIES is a special attribute of request, and its value is the name of the cookie from which you want to read data. Since there can be multiple cookies, you can change this value many times according to the type of cookie you want to store.

[Django Cookies and Cookies Handling - How to Create Cookies in Django - DataFlair \(data-flair.training\)](#)

Django Cookies Handling

Read Cookies from request

Cookies pass to the particular website every time the client makes a request. Therefore, server every time receives a cookie alongside the request. Django makes it very easy for us to retrieve data from a cookie.

2. Using request.COOKIES.get()

Syntax: `COOKIES.get('cookie_name', 'value')`

Django also provides a method to get the desired value from the cookie. You can directly access that value using get method over request object.

[Django Cookies and Cookies Handling - How to Create Cookies in Django - DataFlair \(data-flair.training\)](#)

Django Cookies Handling

Deleting a Cookie in Django

Django provides you with easy methods for deleting cookies.

The **`delete_cookie()`** takes in the name of the cookie to be deleted, and this method is associated with the response object.

[Django Cookies and Cookies Handling - How to Create Cookies in Django - DataFlair \(data-flair.training\)](#)

Django Intermediate Level Topics

- Django Admin Interface
- Django Database
- Django Redirect
- Django Cookies Handling
- **Django Form Handling and Validation**
- Django File Upload
- Django Static File Handling
- Django Bootstrap
- Django CRUD

Django Form Handling and Validation

These days websites have become highly personalized resulting to some awesome applications for users. This personalization is an impact of user-data on the server. And, that data is highly valuable for you as it is used to provide a better user experience and more functionality. But, it also has some serious security consequences and legal issues. Therefore, when designing any user-input, keep in mind that user-data is highly valuable.

Django Form Handling and Validation

Forms are a collection of HTML elements to take input from the user. In simple words, whenever you give any kind of instruction or input on web, that is known as a form.

The input elements come inside **<form></form>** tags. It is collectively called as **HTML form**. All the websites use forms each having different use-cases.

Some daily examples of forms can include:

- Google search
- Facebook posts and stories
- Online registrations
- Online Quizzes, tests, etc

Django Form Handling and Validation

Forms have become an essential part of web applications.

We use them to structure and transmit user-data over **HTTP**. They provide all kinds of input options to be taken from the user.

Django Form Handling and Validation

HTTP is an acronym for **HyperText Transfer Protocol**. It is the set of rules which all the computers on a network follow to transmit data to other computers. All client/server architectures are based on this protocol.

There are different methods to transmit data over HTTP. The 2 very popular methods among all of them are **GET** and **POST**.

Django Form Handling and Validation

1. GET

We use this method for user-data which will not be changing the state of the website. **GET** is a method of transmitting form data via URL.

Google search is the best example of the same. We use this method for transmitting data which doesn't make changes in the database.

GET method comes in handy when we want to bookmark pages as our search query becomes part of the URL.

GET method is often limited by the size of the URL. Anything more than URL size cannot be transmitted. That means we can only send text requests.

Django Form Handling and Validation

2. POST

This method is used for sensitive data and it changes the state of the website. POST is a method of transmitting data as collective structure over HTTP. It is sent as a different request from client alongside URL request, like cookies.

POST data is not a cookie, it is just a different type of resource. This method is more secure than getting because the data is not in URL. The POST method also has a greater size limit, allowing for file uploads.

This method is used on all the registration forms and quizzes.

Django Form Handling and Validation

HTML Forms

The `<form>` tag has two important attributes to be specified:

1. Action

This attribute specifies where to submit the form data. It can be left blank but in that case, the data will be sent to the same URL. Otherwise, you can specify a URL. This attribute just tells the form where to submit the data.

Django Form Handling and Validation

HTML Forms

2. Method

This attribute specifies how to submit data. There are GET and POST values for the same. You should use capital values for this attribute.

They will function as explained in the previous section.

Django Form Handling and Validation

When a form is rendered in Django, these steps are involved:

1. Render the Form in its Initial State
2. Receive the Form Data & Validate the Information
3. If the Received Data is not Valid, Response is a Bounded Form
4. If the Received Data is Valid, a Success Response will generate

[Django Forms Handling & Django Form Validation - Master the Concept - DataFlair \(data-flair.training\)](#)

Django Form Handling and Validation

Form validation means verification of form data. Form validation is an important process when we are using model forms. When we validate our form data, we are actually converting them to python datatypes. Then we can store them in the database.

This validation is necessary since it adds that layer of security where unethical data cannot harm our database.

Django Form Handling and Validation

NOTE:

There are certain JavaScript Validation methods too, that can be easier to implement, but they are not preferred at all as JavaScript is client-side scripting. If the client browser does not have JavaScript enabled, the data will not be validated. And, it causes some serious issues for developers.

Therefore, data validation on the server side is always necessary. Though, you can have JavaScript validation on the client side.

Django Form Handling and Validation

Different Ways of Form Validation:

1. Validation of a Particular Field
2. Using `is_valid()` - This validation will check for Python-datatypes.
3. Validation using Validators
4. Custom Validators

[Django Forms Handling & Django Form Validation - Master the Concept - DataFlair \(data-flair.training\)](#)

Django Intermediate Level Topics

- Django Admin Interface
- Django Database
- Django Redirect
- Django Cookies Handling
- Django Form Handling and Validation
- **Django File Upload**
- Django Static File Handling
- Django Bootstrap
- Django CRUD

Django File Upload

File uploading is just like any input from the user.

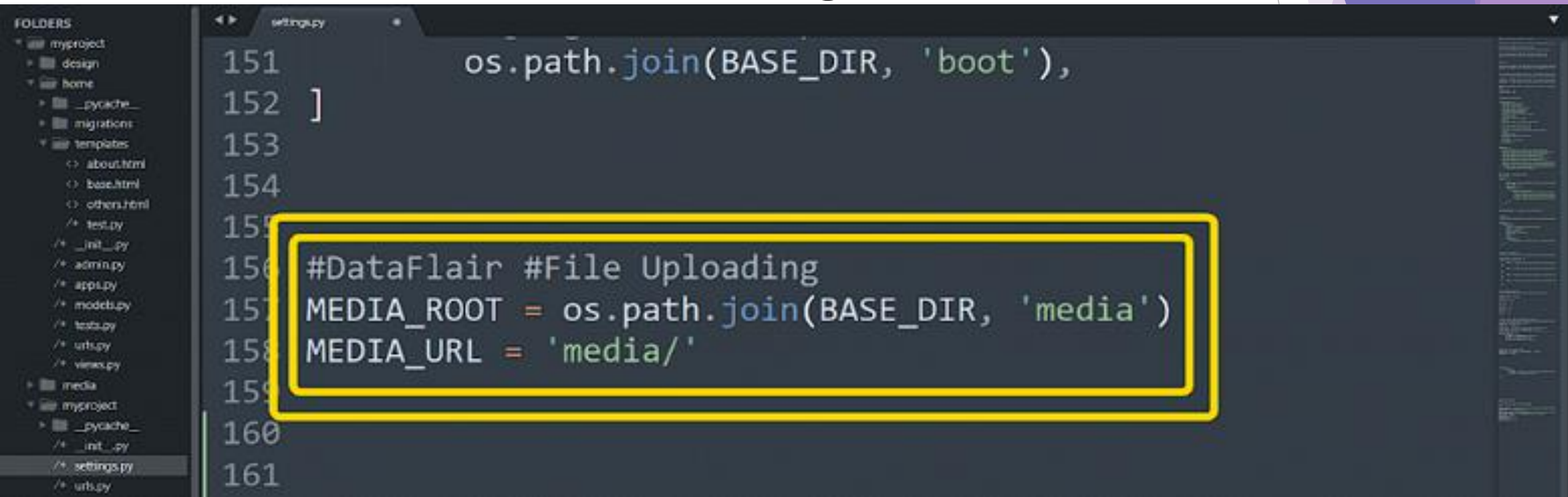
For example – On Facebook, when you update your profile picture. You select a file from your device and the file is then uploaded to the Facebook server. The server stores the file and is then served when anyone visits your profile.

You can say the file is the same as static content on your website.

Django File Upload

Configuring Settings

Before any file upload application, these settings are necessary. Open project's **settings.py** file and add these lines below the **STATIC FILES** settings.



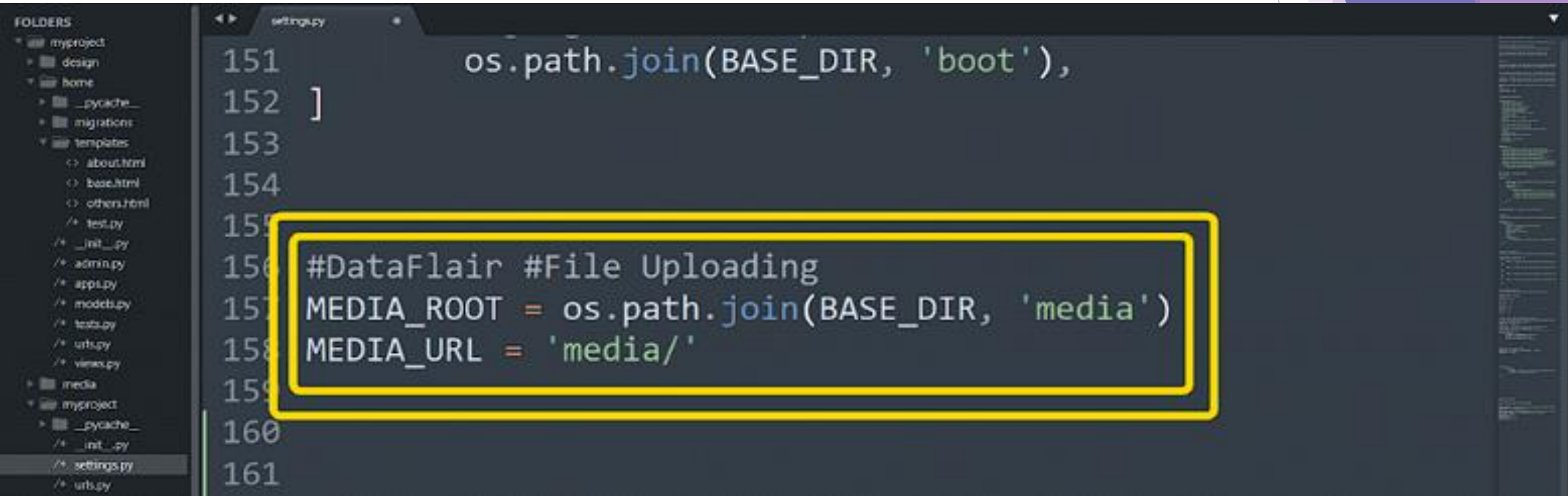
```
FOLDERS
* myproject
  design
  home
    __pycache__
    migrations
    templates
      about.html
      base.html
      others.html
    test.py
  __init__.py
  admin.py
  apps.py
  models.py
  test.py
  urls.py
  views.py
  media
  myproject
    __pycache__
    __init__.py
    settings.py
    urls.py

151     os.path.join(BASE_DIR, 'boot'),
152 ]
153
154
155
156 #DataFlair #File Uploading
157 MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
158 MEDIA_URL = 'media/'
159
160
161
```

Django File Upload

Configuring Settings

MEDIA_ROOT is an empty string by default. It takes in the absolute file path of a directory. This directory will store all the files a user uploads in Django.



The image shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like 'myproject', 'design', 'home', 'templates', 'media', and 'myproject'. The code editor shows the 'settings.py' file with the following code:

```
151 os.path.join(BASE_DIR, 'boot'),  
152 ]  
153  
154  
155  
156 #DataFlair #File Uploading  
157 MEDIA_ROOT = os.path.join(BASE_DIR, 'media')  
158 MEDIA_URL = 'media/'  
159  
160  
161
```

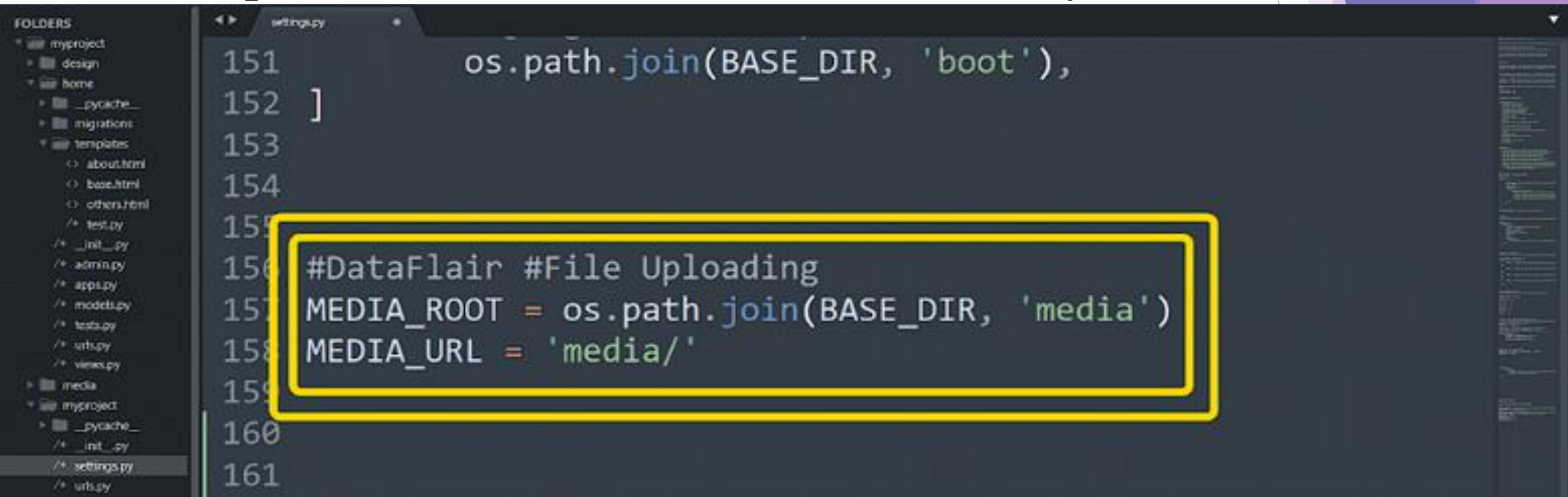
The code for configuring file uploads is highlighted with a yellow box:

```
#DataFlair #File Uploading  
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')  
MEDIA_URL = 'media/'
```

Django File Upload

Configuring Settings

MEDIA_URL works the same way as **STATIC_URL**. This URL is used by Django instead of serving files with an absolute path. When Django will serve an uploaded file media it will be automatically added to the URL.



```
FOLDERS
* myproject
  * design
  * home
    * __pycache__
    * migrations
    * templates
      < about.html
      < base.html
      < others.html
    /* test.py
    /* __init__.py
    /* admin.py
    /* apps.py
    /* models.py
    /* test.py
    /* urls.py
    /* views.py
  * media
  * myproject
    * __pycache__
    /* __init__.py
    /* settings.py
    /* urls.py

151 os.path.join(BASE_DIR, 'boot'),
152 ]
153
154
155
156 #DataFlair #File Uploading
157 MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
158 MEDIA_URL = 'media/'
159
160
161
```

Django File Upload

Note:

MEDIA_URL & STATIC_URL should always have different values. Also, this goes for MEDIA_ROOT & STATIC_ROOT settings too. It was common in the past to use media root to all server static files. With the experiences of developers, this approach has some serious security implications.

It is mandatory to have different values for media settings and static settings.

[How to Upload File in Django - Learn with Easy Steps in just 10 Mins! - DataFlair \(data-flair.training\)](#)

Django Intermediate Level Topics

- Django Admin Interface
- Django Database
- Django Redirect
- Django Cookies Handling
- Django Form Handling and Validation
- Django File Upload
- **Django Static File Handling**
- Django Bootstrap
- Django CRUD

Django Static File Handling

Static files are those files which can not be processed, generated or modified by the server.

Images, JavaScript files, etc are types of content or static files. Static files contain all kinds of file types – from .mpeg, .jpeg to .pdf, etc.



There is a simple concept of working with static files. When a user requests for a webpage, the server will generate the HTML. Then the server will collect all the corresponding static files related to that page. Lastly, this whole data is served.

Django Static File Handling

Benefits of Static Files

- They are **static**: These files don't change until the developer replace them with a new one. Thus, the server just fetches them from the disk, taking a minimum amount of time.
- Static files are **easier to cache**: They don't change and are not modified by the server. That makes the performance faster.
- Static files are **energy efficient**: Static files are fetched from the disk when required. They require no processing which saves the processing overhead and website response becomes fast.

Django Static File Handling

Managing Static Files in Django

A good developer also manages static files. Static file management is an important factor in web development. When the website undergoes frequent UI changes and graphics are updated constantly, this condition requires developers to manage static files.

Django provides us with built-in static file handler. It's a very simple concept and requires some modification in settings.py.

[Django Static Files Handling made Simple - Even your Kids can do it! - DataFlair \(data-flair.training\)](#)

Django Static File Handling

collectstatic command

This command is very useful when the website is in production state. When this command is executed, Django performs these operations:

- It looks for static files in all the directories listed in `STATICFILES_DIRS`
- The static-files are then copied and saved in `STATIC_ROOT` directory.
- When the server is requested for static content, it will fetch a file from `STATIC_ROOT`.
- And, that file will have its URL modified with `STATIC_URL`.

[Django Static Files Handling made Simple - Even your Kids can do it! - DataFlair \(data-flair.training\)](#)

Django Intermediate Level Topics

- Django Admin Interface
- Django Database
- Django Redirect
- Django Cookies Handling
- Django Form Handling and Validation
- Django File Upload
- Django Static File Handling
- **Django Bootstrap**
- Django CRUD

Django Bootstrap

Bootstrap is one of the most popular front-end frameworks out there. It contains some amazing CSS classes for UI development.

Bootstrap has pre-defined CSS files and JavaScript code, which you can link with HTML files. Those CSS files contain classes that can be directly used on HTML elements.

Django Bootstrap

Essential Concepts in Bootstrap

- Bootstrap is an amazing front-end framework. It provides uniform design over a complete website with the least effort.
- Bootstrap is designed in such a way that it adapts between multiple devices. Users can easily access UI made in Bootstrap on mobile, laptops, etc.
- At last, Bootstrap lets you become browser independent. It is compatible with most popular browsers.
- The bootstrap development team keeps their code updated. They also keep it compatible with previous versions of browsers.

Django Bootstrap

Four Major Parts in Bootstrap:

- 1. Layout Components*
- 2. Content*
- 3. Components*
- 4. Utilities*

It is very easy to use Bootstrap in Django. Since Bootstrap is a front-end framework, it completely consists of CSS & JavaScript files. These files are considered static on the server-side.

[Django Bootstrap | An Essential Framework to beat your competitors! - DataFlair \(data-flair.training\)](#)

Django Intermediate Level Topics

- Django Admin Interface
- Django Database
- Django Redirect
- Django Cookies Handling
- Django Form Handling and Validation
- Django File Upload
- Django Static File Handling
- Django Bootstrap
- **Django CRUD**

Django CRUD

CRUD stands for Create, Read, Update & Delete

These are the four basic operations which are executed on Database Models.

1. Read Operation

The ability of the application to read data from the database.

2. Create Operation

The ability of the application to store data in the database.

3. Update Operation

The ability of the application to edit the stored value in the database.

4. Delete Operation

The ability of the application to delete the value in the database.

Applications???

Any questions???

REFERENCES:

- Django Tutorial. (2022). <https://data-flair.training/blogs/django-tutorial/>
- Django Tutorial. (2020). <https://www.geeksforgeeks.org/django-tutorial/>
- Django Tutorial. (2022). <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction>
- Django Tutorial. (n.d.). <https://www.tutorialspoint.com/django/index.htm>
- Django Tutorial. (n.d.). <https://www.w3schools.com/django/index.php>
- Getting started with Django. (n.d.). <https://www.djangoproject.com/start/>