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

## Week 14

# Django Advanced Topics

# Django Advanced Topics

- Django Sessions
- Django Request and Response
- Django Emails
- Django Migrations
- Django ORM
- Django Caching
- Django Admin Customization
- **Django Exceptions & Error Handling**
- AJAX in Django
- Django Web Hosting
- Django CMS
- Django REST Framework
- Django Logging

# Django Exceptions & Error Handling

**Exceptions** are those events in a program occurring of which can lead to undesirable behavior. They are detectable by run time executive or Operating System. Exceptions do not always emerge from errors.

**Errors** are complete halt. If an error occurs, your program will not execute. When the errors like OutOfMemory occurs, your program literally has no memory to execute. Thus, the execution stops.

# Django Exceptions & Error Handling

The programmers can do nothing about errors. Errors can only be caught and an appropriate response can be generated, that's all can be done.

Exceptions are the ones in which we developers actually deal with.

# Django Exceptions & Error Handling

## Django Exceptions

### 1. django.core.exceptions Package

This package provides us with low-level exceptions. It enables us to define new rules for our Django project. We can load models, views in our own defined ways.

This module has use-cases like:

- When you are working on a custom middleware.
- When making some changes to Django ORM.

# Django Exceptions & Error Handling

1. **django.core.exceptions Package**

1.1. **AppRegistryNotReady**

This error occurs when the application model is imported before the app-loading process.

1.2. **ObjectDoesNotExist**

This exception occurs when we request an object which does not exist. It is the base class for all the DoesNotExist Errors.

1.3. **EmptyResultSet**

This error is rare in Django. When we generate a query for objects and if the query doesn’t return any results, it raises this error.

# Django Exceptions & Error Handling

1. **django.core.exceptions Package**

1.4. **FieldDoesNotExist**

This one is clear from its name. When a requested field does not exist in a model, this meta.get_field() method raises this exception.

1.5. **MultipleObjectsReturned**

When we expect a query to return a single response/ object but it returns more than one object, then Django raises this exception.

1.6. **SuspiciousOperation**

It is one of the security classes of Django. Django raises this error when it has detected any malicious activity from the user.

# Django Exceptions & Error Handling

1. **django.core.exceptions Package**

1.7. **PermissionDenied**

This exception is the clearest of all. You must have dealt with this exception while working on static files. Django raises this error when we store our static files in a directory which is not accessible.

1.8. **ViewDoesNotExist**

We all have experienced this one. Websites have a very evolving frontend design and there are frequent modifications which can lead to some faulty urls.

1.9. **MiddlewareNotUsed**

Django is very useful at times. It raises this exception when an unused middleware is present in MIDDLEWARES list.

# Django Exceptions & Error Handling

1. **django.core.exceptions Package**

1.10. **ImproperlyConfigured**

You must have encountered this exception when configuring your first project. This exception is for the main settings.py file.

1.11. **FieldError**

We raise field errors when models have some errors.

1.12. **ValidationError**

We used the validation error in validating the form data.

# Django Exceptions & Error Handling

## Django Exceptions

### 2. URL Resolver Exceptions

This class is a major part of urls.py for defining urls. We import our path function from urls class.

django.urls is one of the core classes of Django without which it might not function. This class also provides some exceptions:

# Django Exceptions & Error Handling

## 2. URL Resolver Exceptions

### 2.1. Resolver404

We raise this exception if the path() doesn't have a valid view to map. The Resolver404 shows us the error page. It is **django.http.Http404** module's subclass.

### 2.2. NoReverseMatch

It is a common occurrence. When we request for a URL which is not defined in our urls-config, we get this error.

# Django Exceptions & Error Handling

## Django Exceptions

### 3. Database Exceptions

We can import these exceptions from django.db module. The idea behind this implementation is:

Django wraps the standard database exceptions. And, when it happens, our Python code can correspond with the database exception. That makes it easy to deal with database exceptions in Python at a certain level.

The exception wrappers provided by Django work in the same way as Python database API.

# Django Exceptions & Error Handling

## 3. **Database Exceptions**

The errors are:

- InterfaceError
- DatabaseError
- DataError
- IntegrityError
- InternalError
- ProgrammingError
- NotSupportedError

We raise these errors when:

- Database is not found
- Interface is not there
- Database is not connected
- Input data is not valid etc.

# Django Exceptions & Error Handling

## Django Exceptions

### 4. Http Exceptions

We used this class in our first views. The HttpResponse is a sub-class of django.http class. The module provides some exceptions and special responses.

### UnreadablePostError

This exception occurs when a user uploads a corrupt file or cancels an upload. In both cases, the file received by the server becomes unusable.

# Django Exceptions & Error Handling

## Django Exceptions

### 5. Transaction Exceptions

A transaction is a set of database queries. It contains the smallest queries or atomic queries. When an error occurs at this atomic level, we resolve them by the django.db.transaction module.

There are errors like TransactionManagementError for all transaction-related problems with the database.

# Django Exceptions & Error Handling

## Django Exceptions

6. **Testing Framework Exceptions**

The django.test package also comes with many exceptions.

7. **Python Exceptions**

Django is a Python framework. Of course, we get all the pre-defined exceptions of Python as well. Python has a very extensive library of exceptions.

Django Exceptions & Error-handling made Easy with this Handy Guide! - DataFlair (data-flair.training)

# Django Advanced Topics

- Django Sessions
- Django Request and Response
- Django Emails
- Django Migrations
- Django ORM
- Django Caching
- Django Admin Customization
- Django Exceptions & Error Handling
- **AJAX in Django**
- Django Web Hosting
- Django CMS
- Django REST Framework
- Django Logging

# AJAX in Django

AJAX has now become an important part of web applications. AJAX also helped Django a lot to increase efficiency and save the time of users. All the popular services out there use it in some way or the other.

Applications like Facebook, Twitter, Instagram, Gmail, Google Maps, Spotify, etc cannot function without AJAX.

# AJAX in Django

**AJAX**

- Asynchronous JavaScript and XML.
- It is a mixture of technologies implemented to solve a very serious problem.
- It utilizes XMLHttpRequest objects to transmit and receive data.

# AJAX in Django

## How AJAX works in Django

AJAX is nothing but a combination of JavaScript and XHR object. The concept is simple:

- JavaScript Code on the client
- The XHR object also contains the URL or name of the call-back function on server.
- The request is handled by the server with a callback function
- At that time, the server processes the request.
- The response is received as success and failure

AJAX in Django - Learn How it Works using jQuery! - DataFlair (data-flair.training)

# Django Advanced Topics

- Django Sessions
- Django Request and Response
- Django Emails
- Django Migrations
- Django ORM
- Django Caching
- Django Admin Customization
- Django Exceptions & Error Handling
- AJAX in Django
- **Django Web Hosting**
- Django CMS
- Django REST Framework
- Django Logging

# Django Web Hosting

Hosting is the part where you can showcase your work to the public. When websites go live, it becomes accessible by the public. People can see and access what you have built.

Django Web Hosting and IDE - An Easy-to-Implement Guide for Beginners - DataFlair (data-flair.training)

# Django Advanced Topics

- Django Sessions
- Django Request and Response
- Django Emails
- Django Migrations
- Django ORM
- Django Caching
- Django Admin Customization
- Django Exceptions & Error Handling
- AJAX in Django
- Django Web Hosting
- **Django CMS**
- Django REST Framework
- Django Logging

# Django CMS

A CMS is a Content Management System. This system lets its user add, modify and delete the content as per user's will. The content we are talking about is digital content. It includes text, images. This article is a perfect example of that kind of content.

# Django CMS

The CMS provides an interface that can be easily used by clients to maintain their websites. The client need not be familiar with backend technologies. A CMS, in short, provides the client-user an interface to easily add, modify and remove the content. Also, it provides the tools to publish the content.

# Django CMS

Django CMS is an Enterprise-grade Management System for content. It was awarded the Best Open Source CMS of 2015. Django utilizes all its features. That itself is very powerful. The extra functionality you get for publishing and content management is awesome.

Django CMS is trusted by many popular organizations. You will be surprised but it is used by NASA, National Geographic, etc.

Django CMS Tutorial - Complete Installation Process & Benefits - DataFlair (data-flair.training)

# Django Advanced Topics

- Django Sessions
- Django Request and Response
- Django Emails
- Django Migrations
- Django ORM
- Django Caching
- Django Admin Customization
- Django Exceptions & Error Handling
- AJAX in Django
- Django Web Hosting
- Django CMS
- **Django REST Framework**
- Django Logging

# Django REST Framework

**Django REST Framework** (DRF) allows developers to rapidly build RESTful APIs.

**API** is an acronym for Application Programming Interface. An API is used by two applications trying to communicate with each other over a network or Internet.

- The API acts as a mediator between Django and other applications. Other applications can be from Android, iOS, Web apps, browsers, etc.
- The API's main task is to receive data from other applications and provide them to the backend. This data is usually in JSON (JavaScript Data Structure) format.

# Django REST Framework

**REST** stands for Representational State Transfer. REST is an architecture on which we develop web services. Web services can be understood as your device connects to the internet.

When you search for anything on Google or watch something on YouTube, these are web services where your device is communicating to a server.

When these web services use REST Architecture, they are called **RESTful Web Services**. These web services use HTTP to transmit data between machines.

# Django REST Framework

A **RESTful API** acts as a translator between two machines communicating over a Web service. This is just like an API but it's working on a RESTful Web service. Web developers program REST API such that server can receive data from applications.

These applications can be web-apps, Android/iOS apps, etc. RESTful APIs today return JSON files that can be interpreted by a variety of devices.

# Django REST Framework

**DRF** is an acronym for Django REST Framework. DRF is a framework built upon the Django Framework. It is not a separate framework. You can say that it is a tool which alongside Django is used to develop RESTful APIs. It increases the development speed. It also addresses various security issues natively.

DRF is an Open Source Software, just like Django. It is actively developed by various MNCs. If you made any profit from this framework then please donate it to keep it open source.

Django REST Framework Tutorial - Feel Blessed!! 'Coz Boss wants you to REST - DataFlair (data-flair.training)

# Django Advanced Topics

- Django Sessions
- Django Request and Response
- Django Emails
- Django Migrations
- Django ORM
- Django Caching
- Django Admin Customization
- Django Exceptions & Error Handling
- AJAX in Django
- Django Web Hosting
- Django CMS
- Django REST Framework
- **Django Logging**

# Django Logging

**Logging**

- a technique or medium which can let us track some events as the software executes.
- Developers are not only responsible for making software but also for maintaining them.
- It tracks every event that occurs at all times. That means instead of the long tracebacks you get much more. This time when an error occurs you can see in which the system was. This method will help you resolve errors quickly.

# Django Logging

The logging is handled by a separate program. That logging program is simply a file-writer. The logger is said to record certain events in text format. The recorded information is then saved in files. The files for this are called **logs**.

The **logging module** is a built-in Python module. It comes preinstalled with Python 3. Logging module is used to keep track of the events that occur as the program runs. That can be extended to any number of programs and thus for software, it is easy to set up and use.

# Django Logging

The logging module is capable of:

- Multithreading execution
- Categorizing Messages via different log levels
- It's much more flexible and configurable
- Gives a more structured information

4 Main Parts of Logging Module:

1. Loggers
2. Handlers
3. Formatters
4. Filters

Django Logging - The Most Easy Method to Perform it! - DataFlair (data-flair.training)

# 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/