Logging file in Django

Software Developer need to get his deployed Django app to write its logs to a file, so that he can look at them later. You can do this by configuring Django's settings as below:

Print works fine when you're using Django's dev web server. It's true! Using "print" works fine locally,

And print in console.

In production with **DEBUG=False**, you won't be able to see your print statements anymore in Django's log output. Log messages will still show up when you're working locally so there's nothing to lose by ditching print for logging.

**How to use logging in Django:**

1. use Python's logging framework to write any log messages that you want to record.
2. Open **view.py** and add below:

# Logging view in Django:

# First import the logging library from Python's standard library:

import os

import logging

# Create a logger for this file or the name of the log level:

logger = logging.getLogger(\_\_file\_\_)

# or Get an instance of a logger:

logger = logging.getLogger(\_\_name\_\_)

1. put below log message in each module and views you defined:

def some\_view(request):

    """

    Example view showing all the ways you can log messages.

    """

    logger.debug("This logs a debug message.")

    logger.info("This logs an info message.")

    logger.warn("This logs a warning message.")

logger.error("This logs an error message.")

    logger.critical("This logs a critical message.")

    try:

        raise Exception("This is a handled exception")

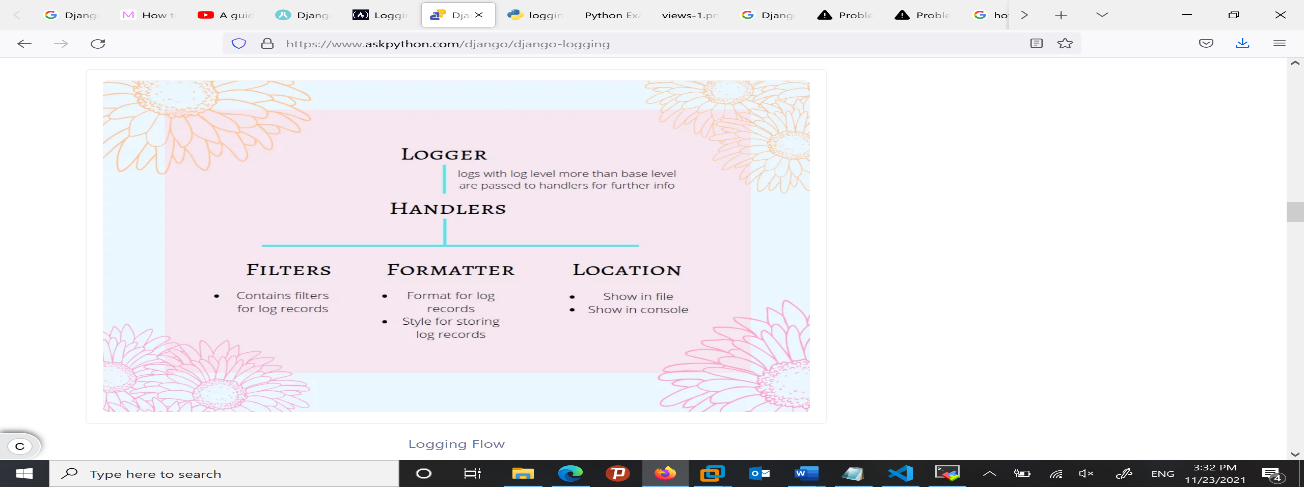
    except Exception:

        logger.exception("This logs an exception.")

    raise Exception("This is an unhandled exception")

    return HttpResponse("this worked")

1. Django comes with Python's built-in logging module to leverage system logging.



1. A Python logging configuration consists of four parts:

Loggers A logger is the entry point into the logging system. Each logger is a named bucket to which messages can be written for processing.

A log-level states the severity of the event that the logger will take in or handle.

| **Log-Level** | **Description** | **Severity** |
| --- | --- | --- |
| **DEBUG** | System information when everything is running fine. | 10 |
| **INFO** | Similar to debug. It generally tells an overview of what the system is executing. | 20 |
| **WARNING** | Involves low-level problems that don’t cause the system to stop. | 30 |
| **ERROR** | This message is serious. This shows the problem may have stopped the operation of the system and needs immediate attention | 40 |
| **CRITICAL** | Most critical message. This is shown when the problem caused the system to stop. | 50 |

**Logging Configuration settings in Django setting.py:**

A log-level states the severity of the event that the logger will take in or handle.

# Configuring logging settings:

# Disabling logging configuration below settings only when need to disable default django logging:

# LOGGING\_CONFIG = None

# logging.config.dictConfig(...)

# Below are configuration logging settings:

LOGGING = {

    'version': 1, # version of logging

    'disable\_existing\_loggers': False,

    # If the disable\_existing\_loggers key in the LOGGING dictConfig is set to True

    # (which is the dictConfig default if the key is missing) then all loggers from the default

    # Configuration will be disabled. Disabled loggers are not the same as removed;

    # The logger will still exist, but will silently discard anything logged to it,

    # Not even propagating entries to a parent logger.

    "handlers": {

        "file": {

            "level": "DEBUG", #This log level describes the severity of the messages that the logger will handle

            "class": "logging.FileHandler",#writes all logging from the django named logger to a local file.

            #"class":"logging.handlers.TimedRotatingFileHandler",# when need to create file every specific period

            #"when": "m", # this specifies the interval hour "h" day "d" , minuites "m" ,seconds "s".

            #'interval': 1, # defaults to 1, only necessary for other values

            #'backupCount': 1, # how many backup file to keep, 1 per day

            "filename": BASE\_DIR / 'django.log', #local logging file name. and path.

            "formatter": "test\_format",

        },

    },

    "loggers": {

        "django": {

            "handlers": ["file"],

            "level": "DEBUG",

            "propagate": True

        },

    },

    "formatters": {

        # log record needs to be rendered as text. Formatters describe the exact format of that text

        "test\_format": {

            "format": (

                u"[%(asctime)s] [%(levelname)-4s] "

                "(%(module)s.%(funcName)s) %(message)s"

            ),

            "datefmt": "%Y-%m-%d %H:%M:%S %p %Z %z",

        },

    },

}