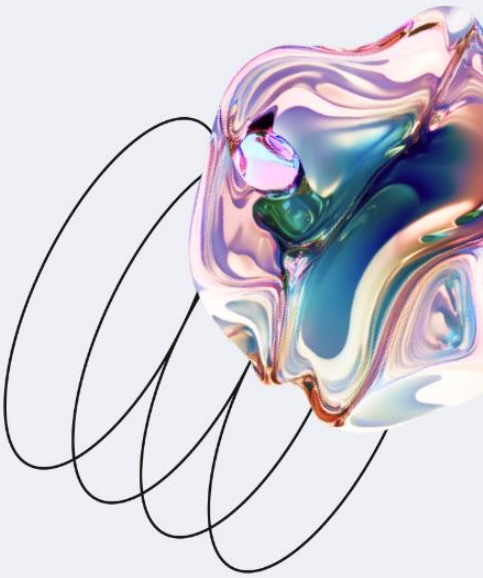
GeekBrains



Лекция 1

Введение в Django

На этой лекции мы

1. Узнаем о фреймворке Django
2. Разберемся в его установке и настройке для первого запуска
3. Изучим структур проекта и работу с ним
4. Узнаем о приложениях как частях проекта
5. Изучим настройки логирования в Django

План лекции

Что такое Django?

Django - это высокоуровневый фреймворк для веб-приложений на языке Python. Он был создан в 2005 году и с тех пор активно развивается и обновляется сообществом разработчиков по всему миру.

Обзор фреймворка Django

Django предоставляет разработчикам множество инструментов и функций для создания веб-приложений, таких как:

- ORM (Object-Relational Mapping) для работы с базами данных
- URL-маршрутизация
- Аутентификация и авторизация пользователей
- Шаблонизация
- Кеширование
- Интернационализация
- Административная панель

Преимущества использования Django

Использование Django имеет множество преимуществ, таких как:

- Быстрая разработка веб-приложений
- Простота и удобство использования
- Высокая производительность
- Безопасность
- Масштабируемость

История развития Django

Django был создан в 2005 году. Первый коммит в июле 2005, а версия 0.90 в ноябре того же года. С тех пор Django активно развивается и обновляется сообществом разработчиков по всему миру. В настоящее время он является одним из самых популярных фреймворков для веб-разработки на языке Python.

История версий

Ниже представлена таблица с версиями Django и датой выхода релиза.

Версия	Месяц и год релиза
0.90	ноябрь 2005
0.91	январь 2006
0.95	июль 2006
0.96	март 2007
1.0	сентябрь 2008
1.1	июль 2009
1.2	май 2010
1.3	март 2011
1.4 LTS	март 2012
1.5	февраль 2013
1.6	ноябрь 2013
1.7	сентябрь 2014
1.8 LTS	апрель 2015
1.9	декабрь 2015
1.10	август 2016
1.11 LTS	апрель 2017
2.0	декабрь 2017
2.1	август 2018
2.2 LTS	апрель 2019
3.0	декабрь 2019

3.1	август 2020
3.2 LTS	апрель 2021
4.0	декабрь 2021
4.1	август 2022
4.2 LTS	апрель 2023
5.0	декабрь 2023

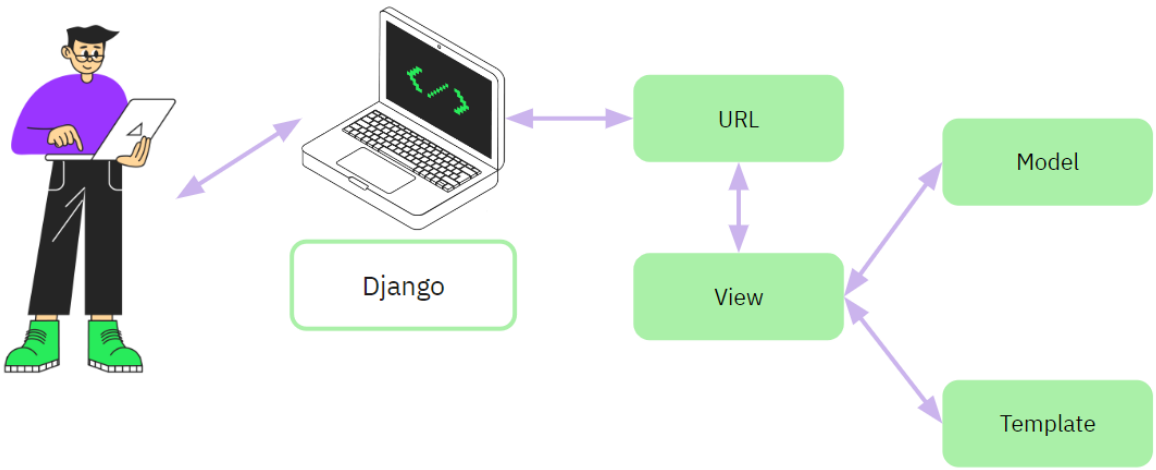
Обратите внимание как долго Django был в нулевой и первой версии. С 2005 по 2014 год обновления выходили хаотично, по мере создания. С 2015 года закрепились месяцы выхода обновлений: апрель, декабрь, август.
Начиная с версии 1.4, стали выходить LTS.
LTS (Long Time Support) — версия, с поддержкой в течение длительного времени (3 года). LTS-версии стабильнее по сравнению с обычными, так как при их выпуске разработчики стараются не экспериментировать со всевозможными новинками.

В декабре 2017 года выходит Django 2.0. Он меняет подход к нумерации версий. С тех пор это тройка x.0, x.1 и x.2. При этом версии x.2 выходят как LTS.

Важно! Даже если сейчас последняя версия имеет номер 5, это не значит, что тот же Django 3.2 более не актуален и изучать его не стоит. Это распространенная ошибка новичка. Большая часть материалов, изучаемых в рамках курса появилась в первых версиях Django и сохраняет свою актуальность до сих пор.

Концепция MVT

Концепция Модель-Представление-Шаблон (MVT) является основой фреймворка Django. Она разделяет приложение на три основных компонента: модель (Model), представление (View) и шаблон (Template).



Модель - это объект, который представляет данные в приложении и отвечает за их хранение и манипуляцию. Модели Django обычно соответствуют таблицам в базе данных.

Представление - это компонент, который отвечает за обработку запросов и формирование ответов на основе данных из моделей. В Django представления обычно реализуются в виде функций или классов.

Шаблон - это компонент, который отвечает за отображение данных на странице. Шаблоны Django используют язык шаблонов, который позволяет вставлять данные из моделей и представлений в HTML-код.

Отдельно стоит упомянуть маршруты. **Маршруты** в Django играют роль посредника между пользователем и представлением. Они определяют, какой URL-адрес должен быть связан с каким представлением. Когда пользователь запрашивает URL-адрес, Django использует маршруты для определения соответствующего представления и передачи запроса ему. Представление обрабатывает запрос и формирует ответ, который затем возвращается пользователю.

Установка и настройка Django

Python помогают решить проблему управления зависимостями проектов. Если у вас есть несколько проектов на Python, то вероятность того, что вам придется работать с разными версиями библиотек или самого Python, очень высока. Но использование виртуальных сред позволяет создавать независимые группы библиотек для каждого проекта, что предотвращает конфликты между версиями и не дает одному проекту повлиять на другой. В Python уже есть

модуль venv для создания виртуальных сред, который можно использовать как в разработке, так и в производстве.

Виртуальное окружение

Создаём виртуальное окружение в Linux или MacOS

```
mkdir project
cd project
python3 -m venv venv
```

Создаём виртуальное окружение в Linux или MacOS

```
mkdir project
cd project
python -m venv .venv
```

Далее активируем виртуальное окружение, чтобы все дальнейшие действия выполнялись внутри него.

```
venv/bin/activate # Linux/MacOS
venv\Scripts\activate # Windows
venv\Scripts\activate.ps1  # Windows PowerShell
```

Установка Django и обязательный компонентов

Для установки Django рекомендуется использовать менеджер пакетов pip. Для этого необходимо открыть терминал и выполнить команду:

```
pip install django
```

Помимо самого фреймворка будет установлено несколько обязательных зависимостей. Впрочем сам Django содержит в себе около 15 пакетов, позволяющих решать большинство задач разработки без установки дополнительных пакетов и модулей.

Создание первого проекта

В Django различаются термины проект и приложение. Проект представляет из себя пакет Python с базовыми настройками. Приложение также является

пакетом. Но он входит в состав проекта. При этом каждый проект может состоять из нескольких приложений. А приложения можно переносить из одного проекта в другой.

Создание проекта

Для создания нового проекта в Django необходимо выполнить команду django-admin startproject <project_name>. Эта команда создаст структуру проекта, которая будет содержать все необходимые файлы и папки для работы с фреймворком.

Внимание! Убедитесь что вы находитесь в нужном каталоге и виртуальное окружение с установленным фреймворком активно.

Пример:

```
django-admin startproject myproject
```

В результате выполнения команды будет создана структура вашего проекта.

Обзор структуры проекта

Структура проекта Django имеет следующий вид:

```
myproject/
    manage.py
    myproject/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

Рассмотрим каждый из созданных файлов.

- manage.py - файл, который используется для управления проектом. С его помощью можно запустить сервер, создать миграции, создать суперпользователя и т.д.
- myproject/ - директория, которая содержит основные файлы проекта.
- __init__.py - файл, который сообщает Python, что директория myproject является пакетом.

- settings.py - файл, который содержит настройки проекта, такие как базы данных, шаблоны, статические файлы и т.д.
- urls.py - файл, который содержит маршруты приложения.
- asgi.py - файл, который используется для запуска проекта в ASGI-совместимых серверах.
- wsgi.py - файл, который используется для запуска проекта в WSGI-совместимых серверах.

Далее мы будем вносить изменения в некоторые из файлов проекта. Но уже в текущем виде он готов к запуску.

Запуск сервера и проверка работоспособности

Для запуска сервера необходимо выполнить команду python manage.py runserver. Эта команда запустит сервер на локальном хосте и порту 8000.

Внимание! Команда должна выполняться из каталога, который содержит файл manage.py. Не забудьте перейти в него командой cd <имя проекта>

Пример:

```
python manage.py runserver
```

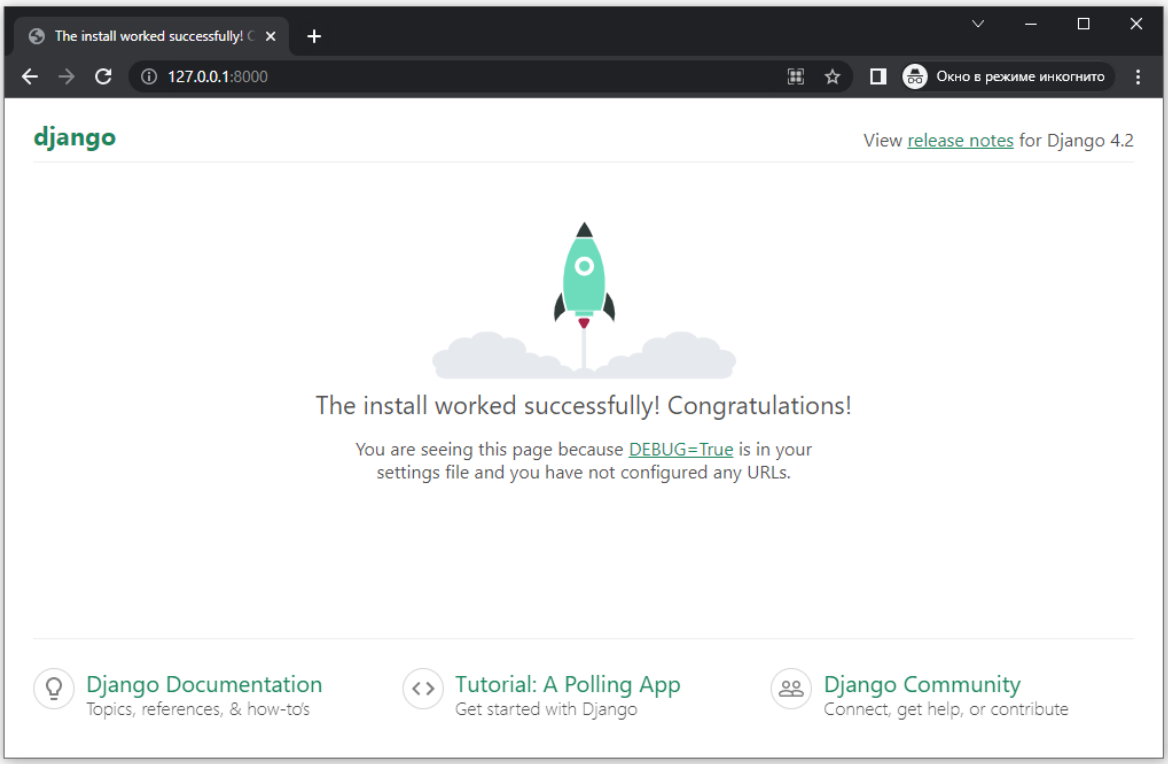
Первый запуск вернёт примерно следующую информацию в консоль:

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly
until you apply the migrations for app(s): admin, auth, contenttypes, se
ssions.
Run 'python manage.py migrate' to apply them.
May 29, 2023 - 10:34:32
Django version 4.2.1, using settings 'myproject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

После запуска сервера можно открыть браузер и перейти по адресу http://localhost:8000/. Если все настроено правильно, то вы увидите страницу "The install worked successfully! Congratulations!" с ракетой.



Особенности встроенного сервера

Сервер Django используется лишь при разработке и тестировании проекта. Он способен раздавать статику в виде изображений, CSS файлов, JavaScript кода и т.п. Так же сервер отслеживает изменения, которые вы вносите в файлы проекта. После каждого такого изменения сервер автоматически перезагружается.

Важно! Встроенный сервер нельзя использовать в продакшене. Он подходит только для разработки проекта.

В процессе работы над проектом мы будем использовать режим отладки. В файле настроек settings.py есть пара строко кода:

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
```

Как вы догадались, отладка помогает нам в создании проекта, но должна быть отключена перед запуском в продакшен.

Предупреждение о непринятых миграциях

Первый запуск сервера выводит предупреждение о том, что в проекте есть непринятые миграции.

```
You have 18 unapplied migration(s). Your project may not work properly
until you apply the migrations for app(s): admin, auth, contenttypes,
sessions.
```

Не стоит переживать за него. Проект будет нормально работать без миграций. Они понадобятся позже, когда начнётся работа с базой данных, создание моделей.

Пара дополнительных параметров runserver

При запуске сервера можно явно заменить порт 8000 на другой. Например так мы воспользуемся портом 8080

```
python manage.py runserver 8080
```

Кроме того можно изменить IP-адрес сервера. Для этого он передаётся вместе с портом

```
python manage.py runserver 0.0.0.0:8080
```

Указав в качестве IP нули мы стали прослушивать все адреса. Если ваш компьютер находится в локальной сети, доступ к проекту можно получить с других устройств этой же сети. Достаточно указать в адресной строке браузера IP-адрес компьютера, на котором запущен сервер. При этом Django может сообщить об ошибке вида:

```
DisallowedHost at /
Invalid HTTP_HOST header: '127.0.0.1:8000'. You may need to add
'127.0.0.1' to ALLOWED_HOSTS.
```

В константу ALLOWED_HOSTS файла settings.py необходимо добавить допустимые адреса в виде списка строк. Например так:

```
ALLOWED_HOSTS = [
    '127.0.0.1',
    '192.168.1.47',
]
```

Важно! Вы всё ещё используете сервер для разработки. Следовательно демонстрируете в локальной сети проект коллегам, а не даёте доступ конечному пользователю.

Создание первого приложения

Создание приложения в Django представляет собой создание отдельного модуля, который будет содержать логику и шаблоны для определенной функциональности. Для создания приложения нужно выполнить команду "python manage.py startapp <app_name>", где <app_name> - название приложения.

Пример:

```
python manage.py startapp myapp
```

Окончание app не является обязательным. Приложение может иметь любое имя, которое отражает его назначение. Но наличии суффикса app позволяет быстрее отличить пакет приложения от другого каталога внутри проекта.

Обзор структуры приложения

Структура проекта после создания приложения имеет следующий вид:

```
myproject/
    myapp/
        migrations/
            __init__.py
        __init__.py
        admin.py
        apps.py
        models.py
        tests.py
        views.py
    myproject/
        __init__.py
        settings.py
        urls.py
        asgi.py
```

```
        wsgi.py
    db.sqlite
    manage.py
```

Рассмотрим каждый из файлов:

- myapp/ - директория приложения
    - migrations/ - директория для хранения миграций базы данных
        - __init__.py - файл, указывающий на то, что директория является пакетом Python
    - __init__.py - файл, указывающий на то, что директория является пакетом Python
    - admin.py - файл для настройки административного интерфейса приложения
    - apps.py - файл для настройки приложения
    - models.py - файл, содержащий модели данных приложения
    - tests.py - файл для написания тестов приложения
    - views.py - файл, содержащий представления (views) приложения
- myproject/ - директория проекта
    - __init__.py - файл, указывающий на то, что директория является пакетом Python
    - settings.py - файл, содержащий настройки проекта
    - urls.py - файл, содержащий маршруты (routes) для обработки URL-адресов
    - asgi.py - файл для запуска ASGI-сервера (Asynchronous Server Gateway Interface)
    - wsgi.py - файл для запуска WSGI-сервера (Web Server Gateway Interface)
- db.sqlite - файл базы данных SQLite
- manage.py - файл для управления проектом Django (запуск сервера, создание миграций и т.д.)

В процессе разработки приложения мы создадим ещё несколько файлов и каталогов. В качестве спойлера, это будут:

- файл маршрутов (urls.py)
- каталог для файлы шаблонов (templates/)

Добавление приложения в проект

Чтобы добавить созданное приложение в проект, необходимо указать его в настройках проекта (файл settings.py). Для этого нужно добавить название приложения в список INSTALLED_APPS.

Внимание! Хорошей привычкой будет делать два действия сразу друг за другом. А именно создавать приложение через startapp и сразу добавляет его в список INSTALLED_APPS.

Учитывая добавленные по умолчанию приложения, константа INSTALLED_APPS будет выглядет так:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myapp',
]
```

За что отвечают перечисленные в списке строки?

- django.contrib.admin - Сайт администратора.
- django.contrib.auth - Система аутентификации.
- django.contrib.contenttypes - Структура для типов контента.
- django.contrib.sessions - Структура сеанса.
- django.contrib.messages - Фреймворк обмена сообщениями.
- django.contrib.staticfiles - Фреймворк для управления статическими файлами.
- myapp - созданное вами приложение

Создание представления в приложении

Следующий пункт работы - создать представление. Так же их называют view или просто “вьюшками”.
Для создания представления нужно определить функцию в файле views.py, которая будет обрабатывать запрос на эту страницу. Пример кода:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world!")

def about(request):
    return HttpResponse("About us")
```

Функция index принимает объект request, который содержит информацию о запросе, и возвращает объект HttpResponse, который содержит ответ сервера. В данном случае мы просто возвращаем текст "Hello, world!".
Вторая функция будет возвращать "About us". Пара простейших представлений созданы. Но работать они пока не будут. Необходимо настроить маршруты.

Настройка путей

Для настройки URL в Django необходимо определить маршруты (routes), которые будут связывать определенные URL с соответствующими представлениями (views) в приложении. Маршруты определяются в файле urls.py, который находится в корневой директории проекта и в директории приложения.
Рассмотрим пример настройки URL в проекте:

Открываем файл urls.py в корневой директории проекта

Файл уже содержит комментарий с описанием трёх способов добавления маршрутов:

- функции представления
- классы представления
- подключение других файлов с настройками URL

Сейчас нам нужен последний пункт. Пропишем добавление файла конфигурации маршрутов из приложения myapp в проект.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapp.urls')),
```

```
]
```

Что делает этот код? Разберём построчно:

- Импортируется модуль admin из пакета django.contrib. С встроенной админкой мы будем работать в рамках курса.
- Импортируется функция path и модуль include из пакета django.urls. Обе функции нужны для формирования url адресов, на которые будет отвечать сервер
- Создается список urlpatterns, который будет содержать маршруты (routes) для обработки URL-адресов.
- Добавляется маршрут для административной панели Django, который будет обрабатывать URL-адрес, начинающийся с префикса "admin/" и передавать управление в модуль admin.site.urls.
- Добавляется маршрут для приложения myapp, который будет обрабатывать пустой URL-адрес и передавать управление в модуль myapp.urls. Маршрут включается с помощью функции include.

Мы добавили одну строку в список urlpatterns. Функция include указывает на несуществующий файл urls.py в каталоге myapp. Если сейчас заглянуть в консоль, работающий сервер сообщит об ошибке, например так:

```
ModuleNotFoundError: No module named 'myapp.urls'
```

Исправим ситуацию с ошибкой.

Создаем файл urls.py в директории приложения

Убедитесь, что вы создаёте файл внутри каталога myapp. Когда в вашем проекте будет несколько приложений, каждое будет иметь собственный urls.py с локальными маршрутами. А urls.py проекта будет включать (include()) их в глобальный список адресов.
Внутри myapp/urls.py пропишем следующий код:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('about/', views.about, name='about'),
]
```

Импортируем уже знакомую функцию path и файл с представлениями из текущего каталога (импорт через точку).
В данном примере мы определяем два маршрута: первый маршрут связывает корневой URL с представлением index, а второй маршрут связывает URL '/about/' с представлением about.
Функция path() принимает два аргумента: первый аргумент - это URL-адрес, а второй - это представление, которое будет обрабатывать запрос на этот URL. Также можно задать имя маршрута с помощью параметра name.

Таким образом, настройка URL в Django позволяет определить маршруты для обработки запросов на определенные URL и связать их с соответствующими представлениями.

Проверка работоспособности

Убедитесь, что сервер работает. Если нет, нужно выполнить команду "`python manage.py runserver`". После этого можно открыть браузер и перейти на страницу http://127.0.0.1:8000/ (по умолчанию), чтобы увидеть текст "Hello, world!". При переходе по адресу http://127.0.0.1:8000/about/ увидим сообщение "About us" из второго представления

В целом, создание приложения и определение представлений - это основа работы с Django. Далее можно добавлять модели, формы, роутинг и другие компоненты для создания полноценного веб-приложения. Всё это будет далее в рамках курса.

Немного подробнее о path() и include()

Прежде чем двигаться дальше, внесём небольшое изменение в написанный код. Разберёмся в том как работают функции path и include из пакета django.urls.
Исправим файл myproject/urls.py. Заменим первый параметр функции path с пустой строки на какой-нибудь текст, например на prefix:

```
from django.contrib import admin
from django.urls import path, include
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('prefix/', include('myapp.urls'))
]
```

Теперь при попытке перейти по адресу http://127.0.0.1:8000/ получит ошибку 404

```
Page not found (404)
Request Method:   GET
Request URL:      http://127.0.0.1:8000/
Using the URLconf defined in myproject.urls, Django tried these URL
patterns, in this order:
admin/
prefix/
The empty path didn't match any of these.
```

У нас нет маршрута для обработки корневого адреса. Попробуем перейти по адресу http://127.0.0.1:8000/prefix/about/ Видим, что представление about отработало свой код и вернула текст.
При вводе адреса в браузер сервер начинает сопоставлять введённый путь с содержимым списка urlpatterns из urls.py проекта. Найдя совпадение, оно отбрасывается и оставшееся часть пути передаётся в urls.py указанный внутри функции include. В нашем примере это myapp/urls.py. Он также просматривает список urlpatterns в поисках совпадений. В нашем примере будет вызвана функция views.about().

Логирование в Django

Рассмотрим, что такое логирование, зачем оно нужно и как его настроить в Django. Также мы рассмотрим примеры использования логирования в Django.

Что такое логирование и зачем оно нужно

Логирование - это процесс записи информации о работе приложения. Обычно это запись в файлы. Хотя можно писать логи и в базы данных, и просто выводить в консоль. Логи позволяют отслеживать работу приложения, выявлять ошибки и проблемы, а также анализировать производительность приложения.

В Django логирование используется для записи информации о запросах, ошибках, отладочной информации и т.д. Логи могут быть полезными для разработчиков, чтобы быстро находить и исправлять ошибки, а также для администраторов, чтобы отслеживать работу приложения и производительность.

Важно! Django не изобретет логирование с нуля. Фреймворк использует модуль logging из стандартной библиотеки Python. Всё, что вы знали о логировании в Python актуально и для логирования в Django.

Конфигурация логирования в Django

Для настройки логирования в Django необходимо изменить файл settings.py. В этом файле определяются параметры логирования, такие как уровень логирования, формат вывода, место хранения логов и т.д .Конфигурация логирования в файле settings.py в Django происходит через словарь LOGGING. Вот пример настройки логирования:

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
        },
        'file': {
            'class': 'logging.FileHandler',
            'filename': '/path/to/django.log',
        },
    },
    'loggers': {
        'django': {
            'handlers': ['console', 'file'],
            'level': 'INFO',
        },
        'myapp': {
            'handlers': ['console'],
            'level': 'DEBUG',
            'propagate': True,
        },
    },
}
```

Давайте рассмотрим каждый параметр:

- version: версия формата конфигурации логирования. В настоящее время используется версия 1.
- disable_existing_loggers: если значение равно True, то все существующие логгеры будут отключены. Если значение равно False, то существующие логгеры будут продолжать работать.
- handlers: определяет, какие обработчики будут использоваться для записи логов. Обработчики могут быть консольными или файловыми.
- loggers: определяет, какие логгеры будут использоваться для записи логов. Логгеры могут быть определены для фреймворка Django или для конкретного приложения.

Для каждого обработчика и логгера можно указать следующие параметры:

- class: класс, который будет использоваться для записи логов. В нашем примере мы используем классы StreamHandler и FileHandler для записи логов в консоль и файл соответственно.
- filename: путь к файлу, в который будут записываться логи. В нашем примере мы записываем логи в файл /path/to/django.log.

  Внимание! Каталог path/ и вложенный в него каталог to/ необходимо создать самостоятельно. Либо исправьте значение на django.log, чтобы создать файл логов в корневой директории проекта.

- level: минимальный уровень логирования, который будет записываться. В нашем примере мы указали уровень INFO для логгера django и уровень DEBUG для логгера myapp.
- propagate: если значение равно True, то сообщения будут передаваться родительским логгерам. Если значение равно False, сообщения не будут передаваться родительским логгерам.

После перезапуска сервера логирование начнёт работать в рамках Django. Перейдите по существующим адресам. Теперь загляните в файл django.log и сравните его содержимое с информацией в консоли.

Примеры использования логирования в Django

Для использования логирования в Django необходимо импортировать модуль logging и создать объект логгера.
Пример двух view функций Django с использованием логирования:

```python
import logging

from django.http import HttpResponse


logger = logging.getLogger(__name__)

def index(request):
    logger.info('Index page accessed')
    return HttpResponse("Hello, world!")

def about(request):
    try:
        # some code that might raise an exception
        result = 1 / 0
    except Exception as e:
        logger.exception(f'Error in about page: {e}')
        return HttpResponse("Oops, something went wrong.")
    else:
        logger.debug('About page accessed')
        return HttpResponse("This is the about page.")
```

В первой функции index мы просто записываем в лог информацию о том, что страница была запрошена. Мы используем логгер info. Сообщение будет выведено на консоль.
Во второй функции about мы используем логгер exception и debug. Если код выполняется без ошибок, мы записываем в лог информацию о том, что страница была запрошена. Если происходит ошибка, мы записываем в лог информацию об ошибке с помощью метода exception, который автоматически добавляет трассировку стека. Затем мы возвращаем пользователю сообщение об ошибке.
Обратите внимание, что ошибки происходят внутри приложения myapp и не сохраняются в файле логов. Наши настройки прописывают сохранение в файл только логов уровня приложения. Поменяем местами параметр file:

```
        'django': {
            'handlers': ['console'],
            'level': 'INFO',
        },
        'myapp': {
            'handlers': ['console', 'file'],
            'level': 'DEBUG',
            'propagate': True,
```

```
        },
```

Теперь мы видим логи уровня приложения и в консоли, и в файле.

Добавление форматирования в логи

Как и в случае со стандартным logging из Python мы можем прописать формат вывода сообщений. Например такая строка добавляет вывод уровня предупреждения и само сообщение

```
{'format': '%(levelname)s %(message)s'}
```

Очень напоминает старндартную работу с логированием, но через словарь. Рассмотрим модификацию нашего словаря LOGGER с добавлением форматированного вывода

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'verbose': {
            'format': '{levelname} {asctime} {module} {process} {thread}
{message}',
            'style': '{',
        },
        'simple': {
            'format': '%(levelname)s %(message)s'
        },
    },
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
            'formatter': 'verbose',  # добавлен параметр formatter
        },
        'file': {
            'class': 'logging.FileHandler',
            'filename': './log/django.log',
            'formatter': 'verbose',  # добавлен параметр formatter
        },
    },
    'loggers': {
        'django': {
            'handlers': ['console', 'file'],
            'level': 'INFO',
        },
```

```
        'myapp': {
            'handlers': ['console', 'file'],
            'level': 'DEBUG',
            'propagate': True,
        },
    },
}
```

Ключ 'formatters' добавляет пару вариантов форматирования. При этом второй вариант, simple использует классическую строку форматирования через %. Первый вариант использует форматирование через фигурные скобки, что явно прописано в паре 'style': '{',.
Далее мы используем созданные ранее обработчики console и file, но добавляем в них строку 'formatter': 'verbose',. Теперь вывод логов происходит по заданному нами формату.
Освежить знания по логированию вы всегда можете на странице официальной документации Python https://docs.python.org/3/library/logging.html

Выводы о логировании

Логирование позволяет нам отслеживать работу приложения и находить ошибки. Мы можем задавать различные уровни логирования для разных частей приложения и выбирать, какие сообщения будут записываться в файл или выводиться на консоль. Если происходит ошибка, мы можем быстро найти ее в логах и исправить.
Логирование - это важный инструмент для отладки и анализа работы приложения, поэтому рекомендуется использовать его в своих проектах.

Вывод

На этой лекции мы:

1. Узнали о фреймворке Django
2. Разобрались в его установке и настройке для первого запуска
3. Изучили структур проекта и работу с ним
4. Узнали о приложениях как частях проекта
5. Изучили настройки логирования в Django

Домашнее задание

1. Для закрепления материалов лекции попробуйте самостоятельно набрать и запустить демонстрируемые примеры.