

## Лекционное занятие

**Тема:** Пример создания веб-сайта на Django.

**Цель:** Научиться создавать прототип сайта при помощи Django, запускать и останавливать сервер для разработки приложения, создавать модели данных для веб-приложения, использовать административную панель Django для управления сайтом.

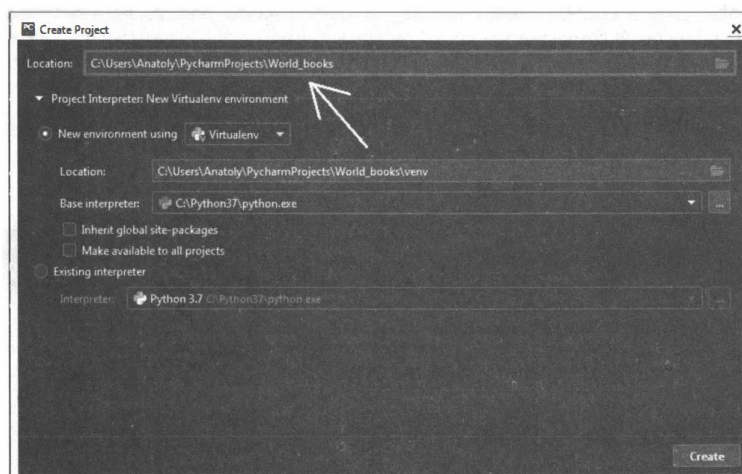
**Оборудование:** персональный компьютер, IDE PyCharm, Visual Studio Code, методические рекомендации к проведению лекционных занятий, Литература: Дронов В. А. Django. Практика создания веб-сайтов на Python - Санкт-Петербург : БХВ-Петербург, 2023. - 800 с., Постолиит А. Python, Django и PyCharm для начинающих – - Санкт-Петербург : БХВ-Петербург, 2021. – 464 с., METANIT.COM: сайт о программировании. - URL: <https://metanit.com/sharp/> – режим доступа: свободный, инструкционная карта для проведения лекционного занятия.

**Создание структуры сайта при помощи Django.** Создадим сайт, которому дадим условное название «Мир книг». Это может быть интернет-ресурс, который дает доступ к скачиванию электронных копий книг, или книжный интернет-магазин, или электронная библиотека, в которую можно записаться и подбирать, и заказывать там себе книги в режиме on-line. Как можно догадаться, цель создания такого достаточно простого сайта заключается в том, чтобы представить онлайн-каталог книг, откуда пользователи смогут загружать доступные книги и где у них будет возможность управлять своими профилями.

На первом шаге мы создадим каталог книг, в котором пользователи смогут только просматривать доступные книги. Это позволит нам изучить операции, которые присутствуют при создании практически любого сайта, где осуществляется взаимодействие с базами данных, - чтение и отображение информации из БД.

По мере развития проекта на сайте будут задействованы более сложные функции и возможности Django. Например, мы сможем расширить функционал сайта, позволив пользователям резервировать книги, покажем, как использовать формы для ввода информации в Бд, как реализовать авторизацию пользователей и т. п.

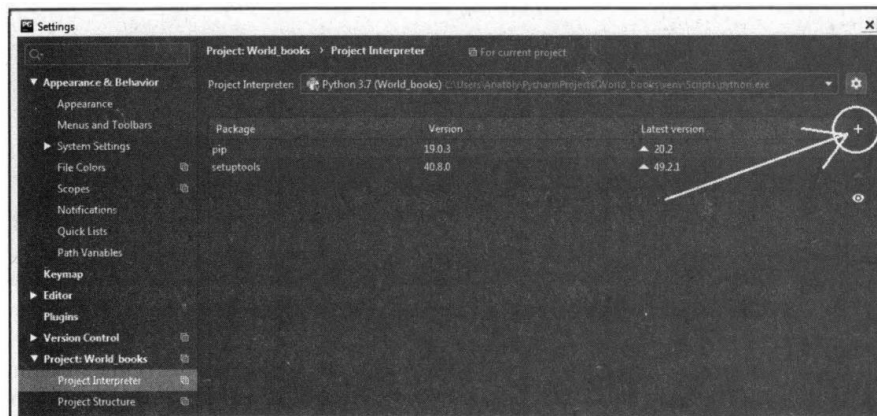
Итак, приступим к созданию «скелета» нашего сайта. Запустите приложение PyCharm и создайте новый проект с именем World\_books. Для этого в главном меню приложения выполните команду File | New Project, после чего откроется окно, в котором введите имя нашего проекта: World\_books.



**Создание нового проекта World\_books в окне приложения PyCharm**

Теперь загрузите веб-фреймворк Django, для чего в главном меню PyCharm выполните команду File / Settings / Project: World\_books / Project Interpreter и в открывшемся окне щелкните мышью на значке

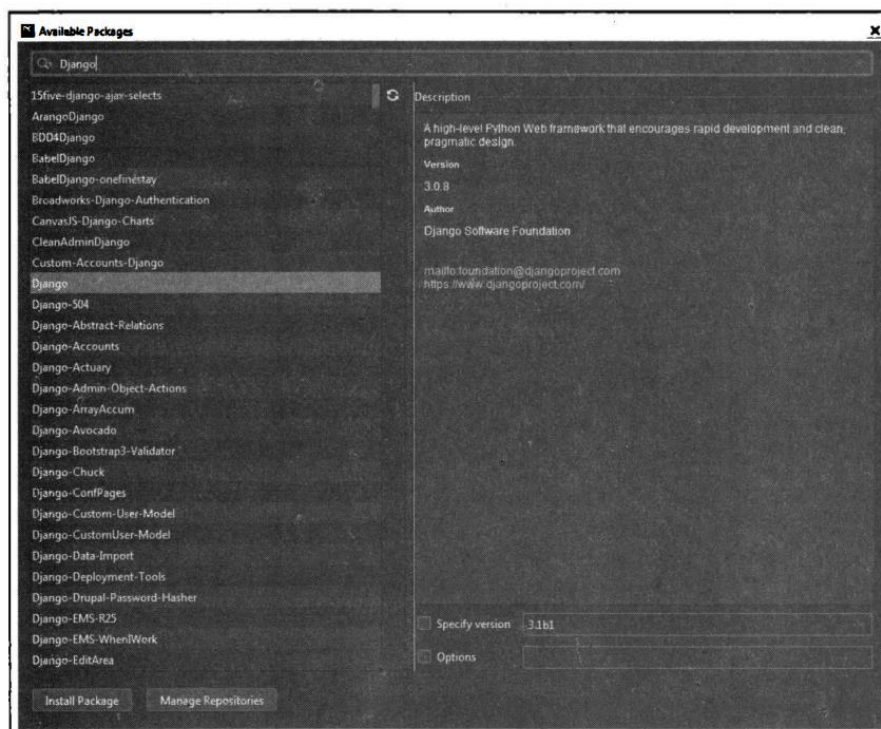




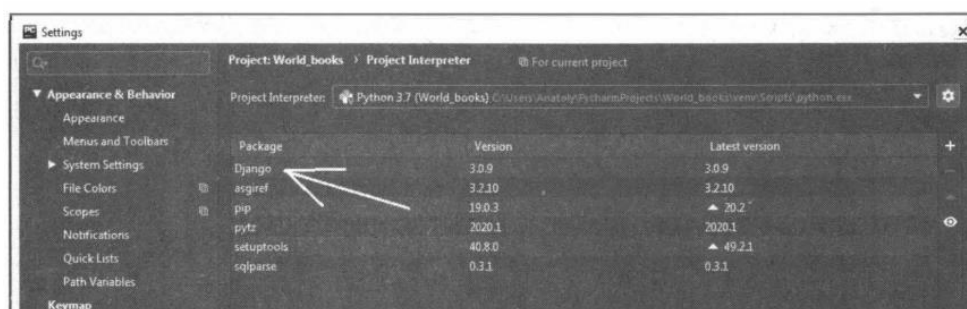
## Окно PyCharm для загрузки дополнительных модулей и библиотек

Здесь можно либо пролистать весь список и найти библиотеку Django, либо набрать наименование этой библиотеки в верхней строке поиска, и она будет найдена в раскрывшемся списке.

Нажмите на кнопку Install Package, и выбранная библиотека Django будет добавлена в ваш проект. Для завершения этого процесса нажмите кнопку ОК в окне Settings.



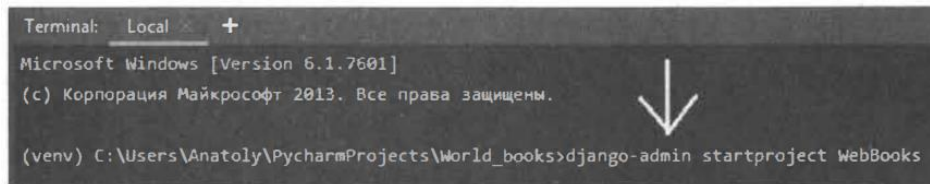
## Поиск библиотеки Django в списке доступных библиотек



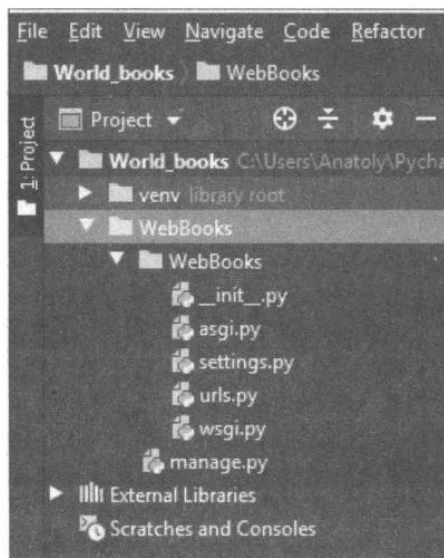
## Библиотека Django в списке доступных библиотек проекта world\_books

Итак, мы создали проект `World_books`, но это пока только проект приложения `PyChann`, который ещё не является проектом `Django`. Поэтому войдите в окно терминала `PyChann` и создайте новый проект `Django` с именем `WebBooks`. Для этого в окне терминала нужно выполнить следующую команду:  
*`django-admin startproject WebBooks`*

**Рис. 8.5. Создание нового проекта Django с именем WebBooks в окне терминала PyCharm**



В результате выполнения этой команды в папке `World_books` проекта `PyChann` будет создана новая папка `WebBooks` проекта `Django`.



**Структура файлов проекта Django с именем WebBooks**

Как можно видеть, папка проекта `WebBooks` включает одну вложенную папку `WebBooks` - это ключевой каталог нашего проекта, в котором находятся следующие файлы:

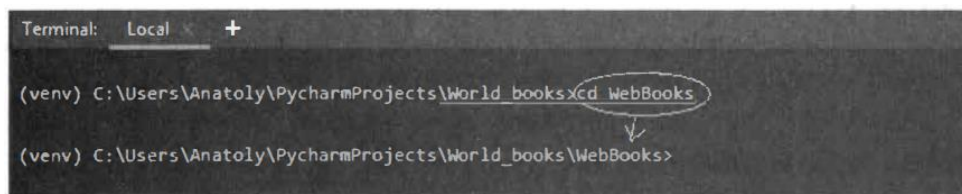
- `settings.py` - содержит все настройки проекта. Здесь мы регистрируем приложения, задаём размещение статичных файлов, делаем настройки базы данных и т. п.);
- `urls.py` - задаёт ассоциации интернет-адресов (URL) с представлениями. Этот файл может содержать все настройки URL, но обычно его делят на части - по одной на каждое приложение, как будет показано далее;
- файлы `asgi.py` и `wsgi.py` используются для организации связи между Django приложением и внешним веб-сервером. Мы задействуем эти файлы позже, когда будем рассматривать развёртывание сайта в сети Интернет.

Кроме вложенной папки `WebBooks`, в головной папке проекта `WebBooks` имеется также файл `manage.py`. Это скрипт, который обеспечивает создание приложений, работу с базами данных, запуск отладочного сервера. Именно этот скрипт мы и будем использовать на следующем шаге для создания приложения.

Итак, создадим приложение с именем `catalog`. Для этого сначала необходимо из приложения `PyCharm` (внешняя папка `World_books`) перейти в приложение `Django` - войти во вложенную папку `WebBooks`,

в которой находится скрипт `manage.py`. Для этого в окне терминала PyCharm выполните следующую команду:

*`cd WebBooks`*

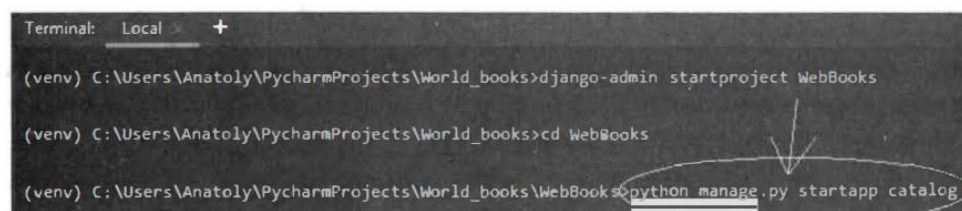


```
Terminal: Local +  
(venv) C:\Users\Anatoly\PycharmProjects\World_books>cd WebBooks  
(venv) C:\Users\Anatoly\PycharmProjects\World_books\WebBooks>
```

### Переход в папку с именем WebBooks проекта Django

Вот теперь, находясь в папке проекта Django с именем WebBooks (той, где находится скрипт `manage.py`), создадим приложение `catalog`. Для этого выполним следующую команду:

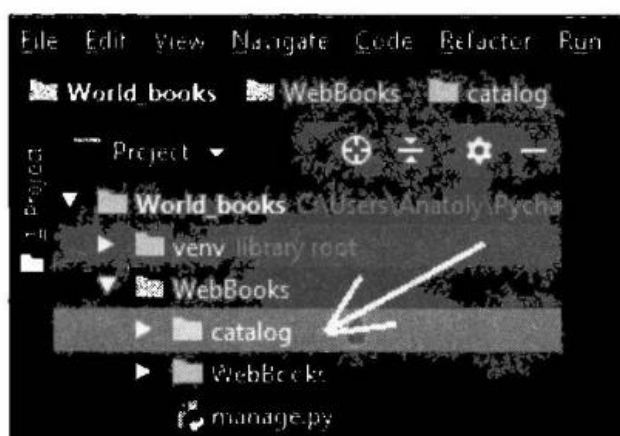
*`python manage.py startapp catalog`*



```
Terminal: Local +  
(venv) C:\Users\Anatoly\PycharmProjects\World_books>django-admin startproject WebBooks  
(venv) C:\Users\Anatoly\PycharmProjects\World_books>cd WebBooks  
(venv) C:\Users\Anatoly\PycharmProjects\World_books\WebBooks>python manage.py startapp catalog
```

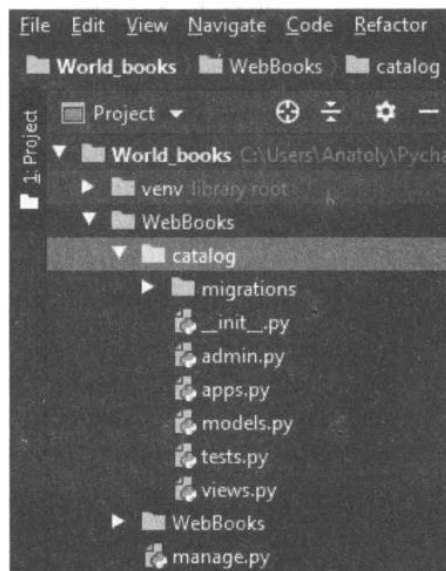
### Создание в проекте Django с именем WebBooks приложения catalog

В результате в папке верхнего уровня WebBooks появится новая папка с именем `catalog` - в ней и будут содержаться файлы нашего приложения `catalog` (рис. 8.10).



### Папка catalog в проекте Django с именем WebBooks





### Содержимое папки catalog приложения catalog

Эти файлы предназначены для выполнения следующих функций:

- папка migrations - хранит миграции (файлы, которые позволяют автоматически обновлять базу данных по мере изменения моделей данных);
- «пустой» файл `__init__.py` - его присутствие позволяет Django и Python распознавать папку catalog как папку с модулями Python, что даёт возможность использовать объекты этих модулей внутри других частей проекта;
- файл `admin.py` - содержит информацию с настройками административной части приложения;
- файл `apps.py` - предназначен для регистрации приложений;
- файл `models.py` - содержит описание моделей данных, с которыми будет работать приложение;
- файл `tests.py` - содержит тесты, которые можно будет использовать для тестирования приложения;
- файл `views.py` - содержит контроллеры или представления (views), которые обеспечивают взаимодействие приложения с БД (выборка, добавление, удаление записей) и с запросами пользователя.

Большинство этих файлов уже содержат некоторый шаблонный программный код для работы с указанными объектами. Однако следует отметить, что для нашего проекта ряда необходимых файлов не хватает - в частности, не были созданы папки для файлов шаблонов и статичных файлов. Далее мы создадим эти папки и файлы (они не обязательны для каждого сайта, но будут нужны в нашем примере).

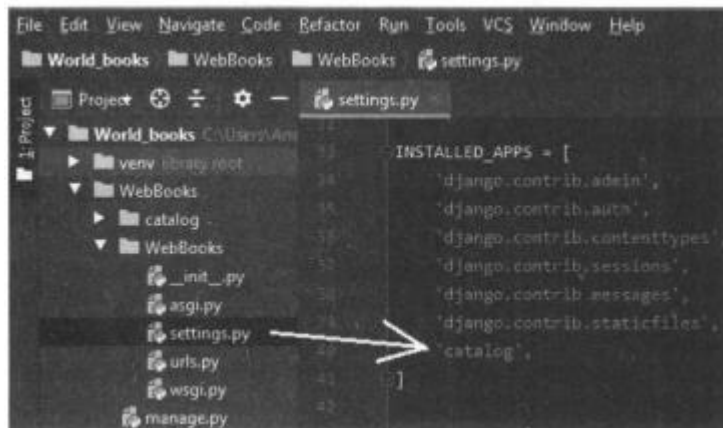
Созданное приложение нужно зарегистрировать в проекте Django. Это необходимо для того, чтобы различные утилиты распознавали его при выполнении некоторых действий (например, при добавлении моделей в базу данных). Приложения регистрируются добавлением их названий в список `INSTALLED_APPS` в файле настроек проекта `settings.py`.

Откройте файл `WebBooks\WebBooks\settings.py`, найдите в нем список зарегистрированных приложений `INSTALLED_APPS` и добавьте наше приложение catalog в конец списка (выделено серым фоном и полужирным шрифтом), как показано на рисунке:

```

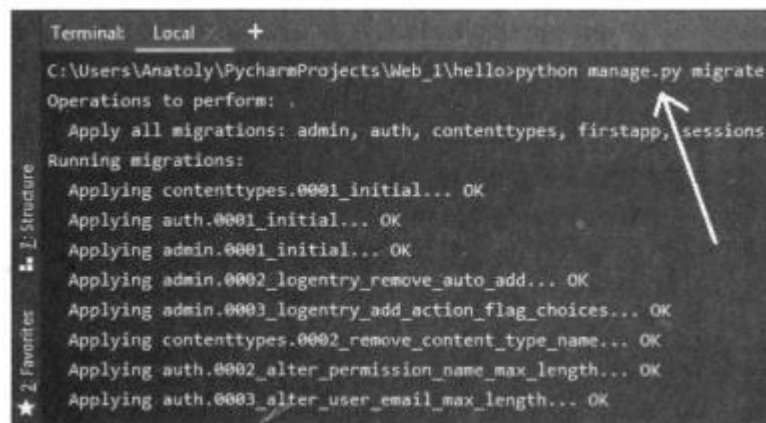
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'catalog',
]

```



### Регистрация приложения catalog в проекте Django

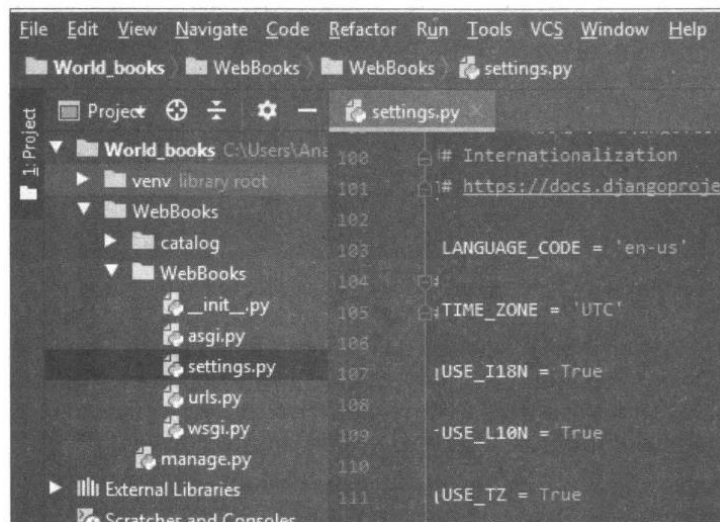
Теперь следует указать базу данных для нашего проекта. По возможности имеет смысл использовать одну и ту же базу данных как в процессе разработки проекта, так и при размещении его в сети Интернет. Это позволит исключить возможные различия в поведении проекта после его публикации. Однако для нашего проекта мы воспользуемся базой данных SQLite, потому что по умолчанию приложение уже настроено для работы с ней. В этом можно убедиться, открыв соответствующий раздел в файле settings.py.



### Настройки проекта для работы с базой данных SQLite

Как можно видеть, к проекту подключён движок БД SQLite, а файл, который содержит сами данные, носит имя db.sqlite3.

Файл settings.py также служит и для выполнения некоторых других настроек (значения их по умолчанию представлены на рисунке:



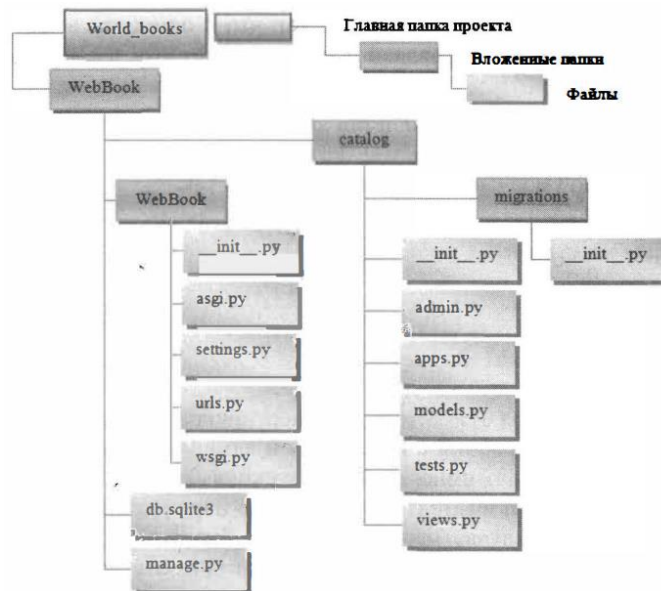
### Настройки проекта в файле settings.py по умолчанию

- LANGUAGE\_CODE - строка, представляющая код языка. Значение по умолчанию: en-us - американская версия английского. Код русского языка для России -ru-ru;
- .TIME\_ZONE - строка, представляющая часовой пояс. Значение по умолчанию: UTC (America/Chicago ). Значение для России: Europe/Moscow;
- USE\_I18N - логическое значение, которое указывает, должна ли быть включена система перевода Django. По умолчанию: True. Если значение USE\_I18N установить в False, то Django не будет загружать механизм перевода - это один из способов повышения производительности приложения;
- USE\_L10N - логическое значение, которое указывает, будет ли включено локализованное форматирование данных. По умолчанию: True (при этом Django будет отображать числа и даты в формате текущей локализации);
- USE\_TZ - логическое значение, которое указывает, будет ли datetime привязан к часовому поясу. По умолчанию: True (при этом Django будет использовать внутренние часы с учётом часового пояса, в противном случае будут задействованы нативные даты по местному времени).

В нашем случае имеет смысл поменять параметры LANGUAGE\_CODE и TIME\_ZONE. В привязке этих параметров к русскому языку и Московскому региону эти настройки будут выглядеть следующим образом:

```
LANGUAGE_CODE = 'ru-ru'
TIME_ZONE = 'Europe/Moscow'
USE_I18N = True
USE_L10N = True
USE_TZ = True
```

В проекте может быть несколько приложений, но мы ограничимся одним. И после всех выполненных нами действий полная структура нашего веб-приложения будет выглядеть так, как представлено на рисунке.



## Полная структура файлов и папок сайта с проектом WebBook и приложением catalog

Теперь определим какие-нибудь простейшие действия, которые будет выполнять это приложение, - например, отправлять в ответ пользователю строку ГЛАВНАЯ страница сайта Мир книг. Для этого перейдем в проекте приложения catalog к файлу views.py, который по умолчанию должен выглядеть так:

```
from django.shortcuts import render
# Create your views here.
```

и изменим этот код следующим образом:

```
from django.shortcuts import render
from django.http import HttpResponse
```

```
# Create your views here.
```

```
def index(request):
    return HttpResponse("Главная страница сайта Мир книг!")
```

Здесь мы импортируем класс HttpResponse () из стандартного пакета django.http. Затем определяем функцию index () , которая в качестве параметра получает объект запроса пользователя request. Класс HttpResponse () предназначен для создания ответа, который отправляется пользователю. Так что с помощью выражения return HttpResponse () мы как раз и отправляем пользователю строку "Главная страница сайта Мир книг! ".

Теперь в проекте Django откроем файл urls.py, обеспечивающий сопоставление маршрутов с представлениями (views), которые будут обрабатывать запросы по этим маршрутам. По умолчанию этот файл выглядит так:

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

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



В первой строке из модуля `django.contrib` импортируется класс `Admin`, который предоставляет возможности работы с интерфейсом администратора сайта. Во второй строке из модуля `django.urls` импортируется функция `path`. Она задает возможность сопоставлять запросы пользователя (определенные маршруты) с функциями их обработки. Так, в нашем случае маршрут `'admin/'` будет обрабатываться методом `admin.site.urls`. То есть если пользователь запросит показать страницу администратора сайта (`'admin/'`), то будет вызван метод `admin.site.urls`, который вернет пользователю HTML-страницу администратора.

Но ранее мы определили функцию `index ()` в файле `views.py`, который возвращает пользователю текстовую строку. Поэтому изменим сейчас файл `urls.py` следующим образом

```
from django.contrib import admin
from django.urls import path
from catalog import views
```

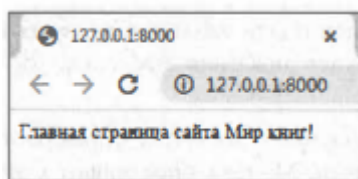
```
urlpatterns = [
    path("", views.index, name='home'),
    path('admin/', admin.site.urls),
]
```

Чтобы использовать функцию `views.index`, мы здесь сначала импортируем модуль `views`. Затем сопоставляем маршрут `('')` - это, по сути, запрос пользователя к корневой странице сайта, с функцией `views.index`. То есть если пользователь запросит показать главную (корневую) страницу сайта `('')`, то будет вызвана функция `index` из файла `views.py`. А в этой функции мы указали, что пользователю нужно вернуть HTML-страницу с единственным сообщением: Главная страница сайта Мир книг!. Соответственно, и маршрут с именем `'home'` также будет сопоставляться с запросом к «корню» приложения.

Теперь запустим приложение командой:

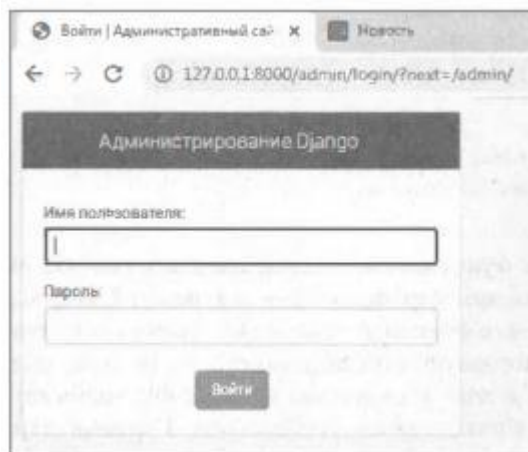
```
python manage.py runserver
```

и перейдем в браузере по адресу `http://127.0.0.1:8000/` - браузер отобразит нам сообщение: Главная страница сайта Мир книг!



### **Первый запуск проекта Webbook с приложением catalog**

Вспомним, что в файле `urls.py` (см. листинг выше) второй строкой прописана возможность запуска встроенного в Django приложения «Администратор сайта» - через маршрут `'admin/'`. Вызовем этот модуль, указав в браузере интернет-адрес (URL): `http://127.0.0.1:8000/admin/`, и получим следующий ответ



### Вызов административной страницы сайта проекта WebBook

Как можно видеть, интерфейс приложения администрирования сайта выведен на русском языке. Это результат того, что в файле `settings.py` мы изменили параметры локализации и установили русский язык (параметр `ru-ru`). К приложению «Администратор сайта» мы вернемся немного позже. А пока можно сделать вывод, что мы создали полноценный «скелет» веб-приложения, которое теперь можно расширять, добавляя новые страницы, представления (views) и модели (models) и устанавливая URL-соответствия между представлениями и моделями данных. Самое время перейти к следующему разделу и начать создавать базу данных, а также писать код, который научит сайт делать то, что он должен делать.

**Разработка структуры моделей данных сайта «Мир книг».** Веб-приложения Django получают доступ и управляют данными через объекты Python, называемые моделями. Модели определяют структуру хранимых данных, включая типы полей, их максимальный размер, значения по умолчанию, параметры выбора, текст меток для форм и пр. Структура и описание модели не зависят от используемой базы данных - она может быть выбрана позднее путем изменения всего нескольких настроек проекта. После выбора какой-либо базы данных разработчику не придется напрямую взаимодействовать с ней (создавать таблицы и связи между ними) - это сделает Django, используя уже разработанную структуру модели. Перед тем как перейти к созданию моделей данных, нам необходимо определить, какую информацию мы будем хранить в БД, сформировать перечень таблиц БД и организовать связи между ними.

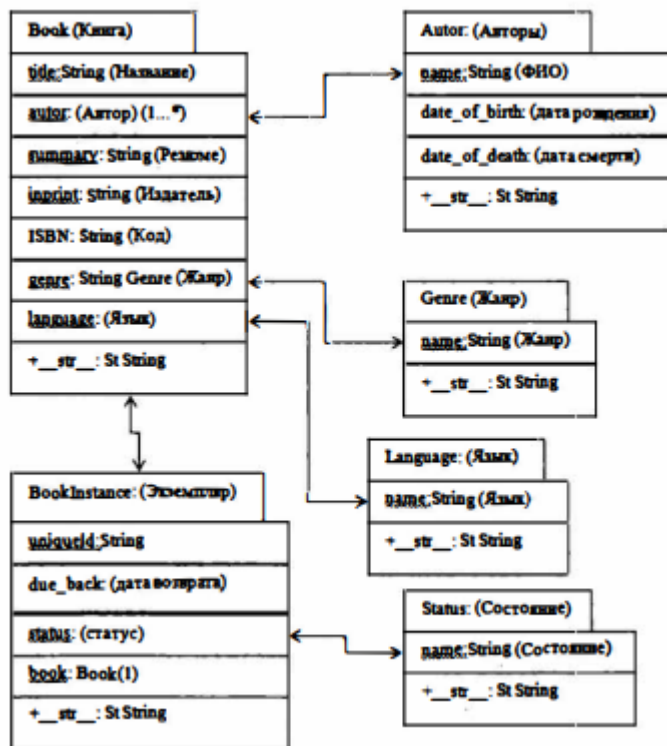
Итак, на нашем сайте мы будем хранить данные о книгах. К этим данным относятся: название книги, ее автор, краткое резюме, язык, на котором написана книга, жанр книги, ее ISBN (International Standard Book Number, международный стандартный книжный номер). Кроме того, необходимо хранить сведения о количестве доступных экземпляров книги, о статусе доступности каждого экземпляра и т. п. Нам также понадобится иметь более подробную информацию об авторе книги, чем просто его имя. Да еще у книги может быть несколько авторов с одинаковыми или похожими именами. Конечно, потребуется возможность сортировки информации на основе названия книги, автора, языка, жанра и других признаков. При проектировании моделей данных имеет смысл создавать самостоятельные модели для каждого объекта. Под объектом мы будем понимать группу связанной информации. В нашем случае очевидными объектами являются: жанры книг, сами книги, их авторы, экземпляры книг, их язык. Следует также определить, какие данные можно и желательно выдавать в виде списка выбора. Это рекомендуется в тех случаях, когда все возможные варианты списка заранее неизвестны или могут впоследствии измениться. Можно выделить следующие данные, которые можно выдавать в виде списков: жанр книги (Фантастика, Поэзия), язык (русский, английский, французский, японский). Как

только будет определен перечень моделей данных (таблицы БД и поля таблиц), нам нужно подумать об отношениях между моделями (между таблицами в БД). Как было показано в предыдущей главе, Django позволяет определять отношения «один-к-одному» (конструктор `OneToOneField`), «один-ко-многим» (конструктор `ForeignKey`) и «многие-ко-многим» (конструктор `ManyToManyField`).

Для нашего сайта мы предусмотрим в БД пять таблиц для хранения информации, и в соответствии с этим нам потребуется сформировать структуру и описание пяти моделей данных. Вот их перечень:

- книги (общие сведения о книгах);
- экземпляры книг (статус конкретных физических книг, доступных в системе);
- автор книги;
- язык, на котором написана книга;
- жанры книг.

Структура этих моделей данных показана на рисунке. Как можно видеть, в ее состав входят перечень таблиц, которые будут созданы в БД, перечень и типы полей, которые будут в каждой таблице, определены также связи между таблицами.



Структура моделей данных сайта «Мир книг»

**Основные элементы моделей данных в Django.** В этом разделе мы вкратце разберемся с тем, как формируются модели данных, рассмотрим некоторые из наиболее важных полей и их аргументы, узнаем, что такое метаданные и как с помощью методов можно управлять данными в моделях. Модели в приложении обычно описываются в файле `models.py`. Они реализуются как подклассы или объекты класса `django.db.models.Model` и могут включать поля, метаданные и методы. В приведенном далее фрагменте кода показан пример типичной модели, имеющей условное название `MyModelName`:

```

from django.db import models

# Типичный класс, определяющий модель, производный от класса Model
class MyModelName(models.Model):

    # Поле (или множество полей)
    my_field_name = models.CharField(max_length=20,
                                     help_text="Не более 20 символов")

    # Метаданные
    class Meta:
        ordering = ["-my_field_name"]

    # Методы
    def get_absolute_url(self):
        # Возвращает url-адрес для доступа к экземпляру MyModelName
        return reverse('model-detail-view', args=[str(self.id)])

    def __str__(self):
        # Строка для представления объекта MyModelName в Admin site
        return self.field_name

```

В этом программном коде определена модель данных (будущая таблица в БД) с именем MyModelName. Таблица БД будет иметь поле my\_field\_name для хранения данных (с количеством символов в имени не более 20). Предусмотрена сортировка данных по полю my\_field\_name и в виде функций заданы два метода. В следующих разделах мы более подробно рассмотрим каждый из элементов внутри модели.

**Поля и их аргументы в моделях данных.** Модель может иметь произвольное количество полей любого типа. Каждое поле представляет столбец данных, который будет сохраниться в одной из таблиц базы данных. Каждая запись (строка в таблице базы данных) будет состоять из значений тех полей, которые описаны в модели. Давайте рассмотрим описание поля из приведенного ранее примера:

```

my_field_name = models.CharField(max_length=20,
                                 help_text="Не более 20 символов")

```

В соответствии с этим описанием таблица БД будет иметь одно поле с именем my\_field\_name, тип поля - CharField. Этот тип поля позволяет хранить в БД строки, состоящие из буквенных и цифровых символов. Система предусматривает проверку содержания этого поля при вводе в него пользователем каких-либо значений в формате HTML. То есть Django не позволит ввести в него недопустимые символы.

Поля могут принимать некоторые аргументы, которые дополнительно определяют, как поле будет хранить данные или как оно может применяться пользователем. В нашем примере полю присваивается два аргумента:

- max\_length = 20 - указывает, что максимальная длина этого поля составляет 20 символов;
- help\_text = "Не более 20 символов" - предоставляет дополнительную текстовую подсказку, чтобы помочь пользователям узнать, какое ограничение наложено на длину этого поля

Метка поля и подсказки будут показаны пользователю через HTML-форму. Имя поля используется для обращения к нему в запросах и шаблонах. У каждого поля также есть метка, значение которой можно задать через аргумент verbose\_name. Метка по умолчанию задается автоматически путем изменения первой буквы имени поля на заглавную букву и замены любых символов подчеркивания пробелом. Например, поле с именем my\_field\_name будет иметь метку по умолчанию My field name.



Поля по умолчанию отображаются в форме (например, на сайте администратора) в том же порядке, в котором они объявляются, хотя этот порядок может быть переопределен в процессе разработки проекта. Поля, описываемые в моделях данных и имеющие разные типы, могут иметь достаточно большое количество общих аргументов:

- `help_text` - предоставляет текстовую метку для HTML-форм (например, на сайте администратора или на форме пользователя);
- `verbose_name` - удобно читаемое имя для поля, используемое в качестве его метки. Если этот аргумент не указан, то Django автоматически сформирует его по умолчанию от имени поля;
- `default` - значение по умолчанию для поля. Это может быть значение или вызываемый объект, и в этом случае объект будет вызываться каждый раз, когда создается новая запись;
- `null` - если аргумент имеет значение `true`, то Django будет хранить пустые значения в базе данных для полей как `NULL`, где это уместно (поле типа `CharField` вместо этого сохранит пустую строку). По умолчанию принимается значение `False`;
- `blank` - если аргумент имеет значение `true`, поле в ваших формах может быть пустым. По умолчанию принимается значение `False`, означающее, что проверка формы Django заставит вас ввести в это поле некоторое значение. Этот аргумент часто используется с аргументом `null = True` - если вы разрешаете ввод пустых значений, вы также должны обеспечить, чтобы база данных могла принять эти пустые значения;
- `choices` - предоставление выбора из вариантов значений для этого поля. Если параметр `choices` задан, то соответствующий виджет формы будет являться полем с вариантами выбора из нескольких значений (вместо стандартного текстового поля);
- `primary_key` - если аргумент имеет значение `True`, то текущее поле будет выступать в качестве первичного ключа для модели (первичный ключ - это специальный столбец базы данных, предназначенный для однозначной идентификации всех разных записей таблицы). Если в качестве первичного ключа не указано ни одно поле, то Django автоматически добавит ключевое поле (`id`) в таблицу данных.

Указанные общие аргументы могут использоваться при объявлении следующих типов полей:

- `CharField` - служит для определения строк фиксированной длины: от короткой до средней. Следует указывать максимальную длину строки `max_length` для хранения данных;
- `TextField` - используется для больших строк произвольной длины. Вы можете указать для поля `max_length`, но это значение будет задействовано только тогда, когда поле отображается в формах (оно не применяется на уровне базы данных);
- `IntegerField` - это поле для хранения значений (целого числа) и для проверки введенных в форму значений в виде целых чисел;
- `DateField` и `DateTimeField` - служат для хранения дат и информации о дате или времени (как `Python datetime.date` и `datetime.datetime`, соответственно). Эти поля могут дополнительно объявлять параметры: `auto_now=True` (для установки поля на текущую дату каждый раз, когда модель сохраняется), `auto_now_add` (только для установки даты, когда модель была впервые создана) и по умолчанию (чтобы установить дату по умолчанию, которую пользователь может переустановить);
- `EmailField` - служит для хранения и проверки адресов электронной почты;
- `FileField` и `ImageField` - служат для загрузки файлов и изображений (`ImageField` просто добавляет дополнительную проверку, является ли загруженный файл изображением). Они имеют параметры для определения того, как и где хранятся загруженные файлы;

- AutoField - это особый тип IntegerField, который автоматически увеличивается. Первичный ключ (id) этого типа автоматически добавляется в вашу модель, если вы явно не укажете его;
- ForeignKey - служит для указания отношения «один-ко-многим» к другой модели базы данных (например, автомобиль определенной модели имеет одного производителя, но производитель может делать много автомобилей разных моделей). Сторона отношения «одиоу» -это модель, содержащая ключ;
- manyToManyField - служит для определения отношения «многие-ко-многим» (например, книга может иметь несколько жанров, и каждый жанр может содержать несколько книг). В нашем приложении для библиотек мы будем использовать этот тип поля аналогично типу ForeignKey, но их можно использовать более сложными способами для описания отношений между группами. Эти типы полей имеют параметр on\_delete, определяющий, что происходит, когда связанная запись удаляется (например, значение models.SET\_NULL просто установило бы значение NULL).

Существует много других типов полей, включая поля для разных типов чисел (большие целые числа, малые целые числа, дробные), логические значения, URLадреса, символы «slugs», уникальные идентификаторы и другие, в том числе «связанные со временем» сведения (продолжительность, время и т. д.). Эти поля мы достаточно подробно описали, когда рассматривали формы Django.

**Метаданные в моделях Django.** Метаданные - это информация о другой информации или данные, относящиеся к дополнительной информации о содержимом или объекте. Метаданные раскрывают сведения о признаках и свойствах, характеризующих какие-либо сущности; позволяющие автоматически искать и управлять ими в больших информационных потоках. К метаданным относятся многие параметры модели, заданные по умолчанию, которые, кстати, можно переопределить. Это, например, имена таблиц, порядок сортировки данных, список индексов, имя поля и т. п.

В Django имеется возможность объявить метаданные о своей модели на уровне самой модели. Для этого нужно объявив класс Meta, как показано в следующем коде:

```
class Meta:
    ordering = ["-my_field_name"]
```

В этом коде показана одна из наиболее полезных функций метаданных - управление сортировкой записей. Порядок сортировки можно задать, указав название поля (или полей). Сам порядок сортировки зависит от типа поля. Например, для текстового поля сортировка будет выполняться в алфавитном порядке, а поля даты будут отсортированы в хронологическом порядке. Прямой порядок сортировки можно заменить на обратный, для этого используется символ минус (-). Согласно порядку, показанному в примере кода, данные текстового поля my\_field\_name (имя клиента) будут отсортированы в алфавитном порядке: от буквы Я до буквы А, поскольку символ ( - ) изменяет порядок сортировки с прямого на обратный.

Если, например, мы решили по умолчанию сортировать книги по двум полям и написали следующий код:

```
class Meta:
    ordering = ["title", "-pubdate"]
```

то книги будут отсортированы по названию согласно алфавиту от А до Я, а затем по дате публикации внутри каждого названия- от самого нового (последнего) издания до самого старого (первого). Другим

распространенным атрибутом является `verbose_name`, или подробное (многословное) имя для класса. Например:

```
from django.db import models

class MyModelName(models.Model):

    my_field_name = models.CharField(max_length=20,
                                     help_text="Не более 20 символов")

    class Meta:
        verbose_name = "Название книги"
```

*Примечание: полный список метаданных доступен в документации по Django*

**Методы в моделях Django.** Модель также может иметь методы, т. е. различные способы манипулирования данными (чтение, обновление, удаление). Минимально в каждой модели должен быть определен стандартный метод класса для Python: `__str__()`. Это необходимо, чтобы вернуть удобно читаемую строку для каждого объекта. Эта строка используется для представления отдельных записей в модуле администрирования сайта (и в любом другом месте, где вам нужно обратиться к экземпляру модели). Часто этот метод возвращает наименование поля из модели, например:

```
def __str__(self):
    return self.field_name
```

Другим распространенным методом, который включается в модели Django, является метод `get_absolute_url()`, который возвращает URL - адрес для отображения отдельных записей модели на веб-сайте. Если этот метод определен, то Django автоматически добавит кнопку Просмотр на сайте на экранах редактирования записей модели (на сайте администратора). Вот типичный шаблон для `get_absolute_url()`:

```
def get_absolute_url(self):
    return reverse('model-detail-view', args=[str(self.id)])
```

Предположим, что требуется использовать URL - адрес, например:

*/myapplication/mymodelname/2*

для отображения отдельной записи модели (где 2 - идентификатор для определенной записи). Чтобы выполнить работу, необходимую для отображения записи, нужно создать URL - карту. Функция `reverse()` может преобразовать ваш URL - адрес (в приведенном примере с именем `model-detail-view`), в URL - адрес правильного формата. Конечно, для выполнения этой работы все равно придется прописать соответствие URL - адреса с представлением (`view`) и шаблоном.

**Методы работы с данными в моделях Django.** Когда классы моделей данных определены, их можно использовать для создания, обновления или удаления записей, для выполнения запросов, а также получения всех записей или отдельных подмножеств записей из таблиц БД. Чтобы создать запись в таблице БД, нужно определить (создать) экземпляр модели, а затем вызвать метод `save()`:

```
# Создать новую запись с помощью конструктора модели
a_record = MyModelName(my_field_name="Книга о вкусной еде")

# Сохраните запись в базе данных.
a_record.save()
```

Если поле `MyModelName` предназначено для хранения названий книг, то в соответствующей таблице БД появится запись с названием этой книги: *Книга о вкусной еде*. Можно получать доступ к полям в

записи БД и изменять их значения. После внесения изменений данных необходимо вызвать метод `save()`, чтобы сохранить измененные значения в базе данных:

```
# Доступ к значениям полей модели
print(a_record.id) # получить идентификатор записи
print(a_record.my_field_name) # получить название книги

# Изменить название книги
a_record.my_field_name="Книга о вкусной и здоровой пище"
a_record.save()
```

Вы можете искать в БД записи, которые соответствуют определенным критериям, используя атрибуты объектов модели. Для поиска и выборки данных из БД в Django служит объект `QuerySet`. Это интегрирующий объект - он содержит несколько объектов, которые можно перебирать (прокручивать). Чтобы извлечь все записи из таблицы базы данных, используйте метод `objects.all()`. Например, чтобы получить названия всех книг из БД, нужно применить следующий код:

```
all_books = Book.objects.all()'
```

Метод `filter()` позволяет отфильтровать данные для `QuerySet` в соответствии с указанным полем и критерием фильтрации. Например, чтобы отфильтровать книги, содержащие в названии слово «дикие») (дикие животные, дикие растения и т. п.), а затем подсчитать их, мы могли бы воспользоваться следующим кодом:

```
wild_books = Book.objects.filter(title__contains='дикие')
n_w_books = Book.objects.filter(title__contains='дикие').count()
```

В этих фильтрах использован формат: `field_name_match_type`, где `field_name` - имя поля, по которому фильтруются данные, а `match_type` - тип соответствия поля определенному критерию (обратите внимание, что между ними стоит двойное подчеркивание). В приведенном коде полем, по которому фильтруются данные, является название книги (`title`) и задан тип соответствия `contains` (с учетом регистра).

В Django можно использовать в фильтрах различные типы соответствия (совпадения). Основные типы соответствия приведены в таблице

№ п/п	Вызов типа соответствия	Назначение типа соответствия	Аналог на языке SQL
1	<code>exact</code>	Точное совпадение с учетом регистра	<code>SELECT ... WHERE id = 14</code>
2	<code>iexact</code>	Точное совпадение без учета регистра	<code>SELECT ... WHERE name ILIKE 'beatles blog'</code>
3	<code>contains</code>	С учетом регистра	<code>SELECT ... WHERE headline LIKE '%Lennon%'</code>
4	<code>icontains</code>	Без учета регистра	<code>SELECT ... WHERE headline ILIKE '%Lennon%'</code>
5	<code>in</code>	Равно одному из элементов списка или кортежа	<code>SELECT ... WHERE id IN (1, 3, 4)</code>
6	<code>gt</code>	Больше чем	<code>SELECT ... WHERE id &gt; 4</code>
7	<code>gte</code>	Больше или равно	<code>SELECT ... WHERE id &gt;= 4</code>
8	<code>lt</code>	Меньше чем	<code>SELECT ... WHERE id &lt; 4</code>
9	<code>lte</code>	Меньше или равно	<code>SELECT ... WHERE id &lt;= 4</code>
10	<code>range</code>	Внутри диапазона	<code>SELECT ... WHERE id &gt; 4 AND id &lt;8</code>

**Основные типы соответствия (совпадения), используемые в Django**



Полный список типов соответствий, которые используются в фильтрах, можно посмотреть в оригинальной документации на Django. В некоторых случаях возникает необходимость фильтровать поле, которое определяет отношение «один-ко-многим» к другой модели. В этом случае вы можете «индексировать» поля в связанной модели с дополнительными двойными подчеркиваниями. Так, например, чтобы выбрать по фильтру книги с определенным жанром, нужно указать имя в поле жанра:

```
books_containing_genre =  
    Book.objects.filter(genre__name__icontains='Фантастика')
```

При использовании этого кода из БД будут выбраны все книги из жанров: фантастика, научная фантастика.

**Литература:** Дронов В. А. Django. Практика создания веб-сайтов на Python - Санкт-Петербург : БХВ-Петербург, 2023. - 800 с., Постолинт А. Python, Django и PyCharm для начинающих – - Санкт-Петербург : БХВ-Петербург, 2021. – 464 с., METANIT.COM: сайт о программировании. - URL: <https://metanit.com/sharp/>– режим доступа: свободный, лекционный материал.