Содержание

Вступительное слово

Django: веб-фреймворк для перфекционистов, которые стараются придерживаться дедлайнов.

Мне нравится этот слоган, потому что бывает, что разработчики легко становятся жертвами перфекционизма, когда им приходится доставлять работоспособный исходный код точно в срок.

Есть много отличных веб-фреймворков, но иногда они требуют от разработчика слишком многого, например правильно структурировать проект, отыскивать нужные плагины и элегантно использовать существующие абстракции.

Django снимает бóльшую часть этой усталости от принятия решений и предоставляет вам гораздо больше. Но этот фреймворк также и большой, так что изучение его с нуля может оказаться непосильным.

Я изучил Django в 2017 году, в лоб, из необходимости, когда мы решили, что он будет ключевой технологией для нашей платформы программирования на Python (CodeChalleng.es). Я заставил себя изучить все тонкости, разрабатывая крупное практическое технологическое решение, которое с момента своего создания служило тысячам начинающих и опытных разработчиков Python.

Где-то в этом путешествии я подобрал раннюю редакцию данной книги. И она оказалось настоящей сокровищницей. Очень близкая нашим сердцам в Pybites, она обучает фрейморку Django, помогая **разрабатывать** интересные приложения для решения практических задач. Мало того, Антонио привносит в работу много реального опыта и знаний, что проявляется в том, как он реализует эти проекты.

И Антонио никогда не упускает возможности представить менее известные функциональности, например оптимизацию запросов к базе данных с помощью Postgres, полезные пакеты, такие как django-taggit, социальную аутентификацию с использованием различных платформ, (модельные) менеджеры, шаблонные теги включения и многое другое.

В нескольких главах этого нового издания он даже добавил дополнительные схемы, изображения и примечания и перешел с jQuery на ванильный JavaScript (ну, не приятно ли?!).

Данная книга не только подробно описывает Django, используя чистые примеры исходного кода, которые хорошо объяснены, но и освещает смежные технологии, которые необходимы любому разработчику Django: Django REST framework, django-debug-toolbar, frontend/JS и, последнее, но не менее важное, Docker.

Что еще более важно, вы найдете целый ряд нюансов, с которыми столкнетесь, и лучших образцов практики, которые вам понадобятся, чтобы стать эффективным разработчиком Django в профессиональной среде.

Найти такой многогранный ресурс, как этот, непросто, и я хочу поблагодарить Антонио за всю ту тяжелую работу, которую он постоянно прилагает, чтобы поддерживать его в актуальном состоянии.

Для меня как разработчика Python, который часто использует Django, книга «Django в примерах» стала моим путеводителем, незаменимым ресурсом, который я хочу иметь под рукой. Всякий раз, когда я возвращаюсь к этой книге, я узнаю что-то новое, даже после того, как прочитал ее несколько раз и использую Django уже целых пять лет.

Если вы отправитесь в это путешествие, то будьте готовы к тяжелой практической работе. Ведь это практическое руководство, так что заварите себе хороший кофе и приготовьтесь полностью погрузиться в кучу исходного кода Django! Но именно так нам лучше всего учиться, верно? :)

– *Боб Белдербос*, соучредитель Pybites

Об авторе

Антонио Меле – соучредитель и технический директор Numero, финтех-платформы, которая позволяет финансовым учреждениям строить, автоматизировать и масштабировать цифровые продукты для управления состояниями. Антонио также является техническим директором Exo Investing, цифровой инвестиционной платформы на базе искусственного интеллекта для рынка Великобритании.

Антонио разрабатывает проекты Django с 2006 года для клиентов из нескольких отраслей. В 2009 году Антонио основал Zenx IT, компанию-разработчик, специализирующуюся на разработке цифровых продуктов. Он работал техническим директором и технологическим консультантом во многих технологических стартапах и руководил коллективами разработчиков, создающими проекты для крупных цифровых компаний. Антонио получил степень магистра компьютерных наук в ICAI – Университете Понтификии Комильяс (Pontificia Comillas), в котором он руководит стартапами, находящимися на ранней стадии. Его отец вдохновил его на увлечение компьютерами и программированием.

О рецензенте

Асиф Сайфуддин – разработчик программного обеспечения из Бангладеш. У него десятилетний профессиональный опыт работы с Python и Django. Помимо работы с различными стартапами и клиентами, Асиф также вносит свой вклад в некоторые часто используемые пакеты Python и Django. За его заслуженный вклад в разработку с открытым исходным кодом ныне он является основным сопровождающим такого ПО, как Celery, oAuthLib, PyJWT и auditwheel. Он также является сосопровождающим нескольких пакетов расширений Django Extensions и инструментария фреймворка Django REST framework. Кроме того, он является членом фонда **Django Software Foundation** (**DSF**) с правом голоса и членом фонда **Python Software Foundation** (**PSF**) с правом участия в разработке/управлении. Для многих молодых людей он является наставником в изучении Python и Django как в профессиональном, так и в личном плане.

*Особая благодарность **Карен Стингел** и **Исмиру Куллолли** за чтение и предоставление отзывов о книге с целью дальнейшего улучшения ее содержимого. Мы очень ценим вашу помощь!*

Предисловие

Django – это веб-фреймворк Python с открытым исходным кодом, который способствует быстрой разработке и чистому, прагматичному дизайну. Он снимает бóльшую часть хлопот, связанных с веб-разработкой, и обеспечивает относительно плавную кривую обучения для начинающих программистов. Django следует философии Python «батарейки включены в комплект», поставляя богатый и разнообразный набор модулей, которые решают распространенные задачи веб-разработки. Простота Django в сочетании с его мощными функциональными возможностями делает его привлекательным как для начинающих, так и для опытных программистов. Django был разработан с учетом простоты, гибкости, надежности и масштабируемости.

В настоящее время Django используется бесчисленными стартапами и крупными организациями, такими как Instagram, Spotify, Pinterest, Udemy, Robinhood и Coursera. Не случайно, что в течение последних нескольких лет в ежегодном опросе разработчиков Stack Overflow разработчики по всему миру неизменно выбирали Django в качестве одного из самых любимых веб-фреймворков.

Эта книга проведет вас через весь процесс разработки профессиональных веб-приложений с помощью Django. Книга посвящена объяснению механизмов работы веб-фреймворка Django путем написания нескольких проектов с нуля. В данной книге содержатся не только наиболее важные аспекты веб-фреймворка, но и объясняется, как применять Django к самым разнообразным реальным ситуациям.

В ней не только рассказывается о Django, но и представлены другие популярные технологии, такие как база данных PostgreSQL, резидентное хранилище Redis, очередь заданий Celery, брокер сообщений RabbitMQ и кеш-сервер Memcached. По ходу чтения книги вы научитесь интегрировать указанные технологии в свои проекты Django, чтобы создавать продвинутые функциональности и разрабатывать сложные веб-приложения.

Книга «*Django 4 в примерах*» проведет вас по всему процессу разработки практических приложений, по ходу дела решая распространенные задачи и внедряя лучшие образцы практики, используя пошаговый подход, которому легко следовать.

Прочитав эту книгу, вы получите хорошее представление о том, как работает Django и как разрабатывать полноценные веб-приложения на Python.

Для кого эта книга предназначена

Данная книга должна послужить руководством для программистов, недавно приступивших к работе с Django. Она предназначена для разработчиков со

знанием Python, которые хотят изучать Django прагматичным образом. Вы можете быть асолютным новичком в Django либо вы уже его немного знаете, но хотите извлечь из него максимальную пользу. Так или иначе, эта книга поможет вам освоить наиболее актуальные области веб-фреймворка, разрабатывая практические проекты с нуля. Вы должны быть знакомы с концепциями программирования, чтобы при чтении понимать излагаемый материал. В дополнение к базовым знаниям Python подразумевается некоторое предварительное знание HTML и JavaScript.

О чем эта книга рассказывает

Данная книга охватывает целый ряд тем разработки веб-приложений с помощью Django. Она поможет вам разработать четыре разных полнофункциональных веб-приложения, работа над которыми ведется на протяжении 17 глав:

- приложение для ведения блога (главы 1–3);
- веб-сайт по управлению визуальными закладками (главы 4–7);
- интернет-магазин (главы с 8 по 11);
- платформу электронного обучения (главы 12–17).

Каждая глава охватывает несколько функциональнх возможностей Django.

Глава 1 «*Разработка приложения для ведения блога*» ознакомит с веб-фреймворком Django посредством создания приложения для ведения блога. Вы создадите базовые модели, представления, шаблоны и URL-адреса блога, чтобы отображать посты блога на страницах. Вы научитесь формировать наборы запросов QuerySet с помощью объектно-реляционного преобразователя Django (ORM) и сконфигурируете встроенный в Django сайт администрирования.

Глава 2 «*Усовершенствование блога за счет продвинутых функциональностей*» научит добавлять в свой блог постраничную разбивку и реализовывать представления на основе классов Django. Вы научитесь отправлять электронные письма с помощью Django, а также обрабатывать и моделировать формы. Вы также реализуете систему комментариев к постам блога.

Глава 3 «*Расширение приложения для ведения блога*» посвящена технике интегрирования сторонних приложений. В этой главе вы ознакомитесь с процессом создания системы тегирования и научитесь формировать сложные наборы запросов QuerySet, чтобы рекомендовать схожие посты. Здесь вы научитесь создавать собственные шаблонные теги и фильтры. Вы также узнаете, как использовать фреймворк карт веб-сайтов и создавать новостную RSS-ленту для своих постов. Вы завершите свое приложение для ведения блога, разработав поисковый механизм, в котором используются возможности полнотекстового поиска PostgreSQL.

Глава 4 «*Разработка социального веб-сайта*» посвящена объяснению техники разработки социального веб-сайта. Вы научитесь использовать встроенный в Django фреймворк аутентификации и расширите модель пользователя конкретно-прикладной моделью профиля. В этой главе вы научитесь

использовать фреймворк сообщений и разработаете конкретно-прикладной бэкенд аутентификации.

Глава 5 «*Реализация социальной аутентификации*» посвящена реализации социальной аутентификации с использованием учетных записей Google, Facebook и Twitter по стандарту OAuth 2 с помощью механизма Python Social Auth. Вы научитесь использовать расширения Django для работы сервера разработки по протоколу HTTPS и адаптировать конвейер социальной аутентификации под конкретно-прикладную задачу автоматизации создания профиля пользователя.

Глава 6 «*Распространение контента на веб-сайте*» научит технике трансформации социального приложения в веб-сайт по управлению визуальными закладками (CMS). Вы определите взаимосвязи многие-ко-многим в моделях и создадите букмарклет JavaScript, который будет интегрирован в ваш проект. В этой главе будет показано, как создавать миниатюры изображений. Вы также научитесь реализовывать асинхронные HTTP-запросы с использованием JavaScript и Django, и вы реализуете бесконечную постраничную прокрутку контента.

Глава 7 «*Отслеживание действий пользователя*» продемонстрирует технику разработки системы подписки для пользователей. Вы дополните свой веб-сайт по управлению визуальными закладками, создав приложение для слежения за потоками активности пользователей. Вы научитесь создавать обобщенные отношения между моделями и оптимизировать наборы запросов QuerySet, поработаете с сигналами и реализуете денормализацию. Вы научитесь использовать меню отладочных инструментов Django Debug Toolbar, чтобы получать соответствующую отладочную информацию. Наконец, интегрируете в свой проект быстрое хранилище данных Redis, чтобы вести подсчет просмотров изображений, и с помощью него создадите рейтинг наиболее просматриваемых изображений.

Глава 8 «*Разработка интернет-магазина*» посвящена обследованию техники создания интернет-магазина. Вы разработаете модели для каталога товаров и создадите корзину покупок, используя сеансы Django. Вы разработаете процессор контекста для корзины покупок и научитесь управлять заказами клиентов. В этой главе вы научитесь отправлять асинхронные уведомления с помощью очереди заданий Celery и брокера сообщений RabbitMQ. Вы также научитесь отслеживать Celery с помощью мониторингового инструмента Flower.

Глава 9 «*Управление платежами и заказами*» посвящена технике интегрирования платежного шлюза в интернет-магазин. Вы выполните интеграцию платежного инструмента Stripe Checkout и будете получать асинхронные уведомления о платежах в своем приложении. Вы реализуете конкретно-прикладные представления на сайте администрирования, а также адаптируете сайт администрирования под конкретно-прикладную задачу экспорта заказов в CSV-файлы. Вы также научитесь динамически создавать счета-фактуры в формате PDF.

Глава 10 «*Расширение магазина*» научит технике создания купонной системы для применения скидок к корзине покупок. Вы обновите интеграцию платежного инструмента Stripe Checkout, чтобы имплементировать купон-

ные скидки, и будете применять купоны к заказам. Вы будете использовать резидентное хранилище Redis для хранения товаров, которые обычно покупаются вместе, и применять эту информацию для разработки механизма рекомендации товаров.

Глава 11 «*Добавление интернационализации в магазин*» покажет, как добавлять интернационализацию в проект. Вы научитесь генерировать файлы перевода и управлять ими, а также переводить строковые литералы в исходном коде Python и шаблонах Django. Вы будете использовать приложение Rosetta, чтобы управлять переводами, и реализуете URL-адреса в зависимости от применяемого языка. Вы научитесь переводить поля моделей с помощью модуля `django-parler` и использовать переводы с помощью ORM-преобразователя. Наконец, создадите локализованное поле формы, используя модуль `django-localflavor`.

Глава 12 «*Разработка платформы электронного обучения*» проведет вас по процессу создания платформы электронного обучения. В ваш проект будут добавлены фикстуры, и вы создадите первоначальные модели для системы управления контентом. Вы будете использовать наследование моделей, чтобы создавать модели данных для полиморфного контента. Вы научитесь создавать конкретно-прикладные модельные поля, разработав поле для упорядочивания объектов. Вы также реализуете представления аутентификации для системы управления контентом.

Глава 13 «*Создание системы управления контентом*» научит технике создания системы управления контентом, используя представления на основе классов и примесных классов. Вы воспользуетесь встроенными в Django группами и системой разрешений, чтобы ограничивать доступ к представлениям, и реализуете наборы форм, дабы редактировать содержимое курсов. Вы также создадите функциональность перетаскивания, чтобы переупорядочивать модули курса и их содержимое с помощью JavaScript и Django.

Глава 14 «*Прорисовка и кеширование контента*» покажет, как реализовывать общедоступные представления для каталога курсов. Вы создадите систему регистрации студентов и будете управлять зачислением студентов на курсы. Вы напишете функциональность прорисовки различных типов контента курсовых модулей. Вы научитесь кешировать контент с помощью кеш-фреймворка Django и конфигурировать кеш-бэкенд Memcached и Redis под свой проект. Наконец, вы научитесь отслеживать Redis с помощью сайта администрирования.

Глава 15 «*Разработка API*» посвящена обследованию техники разработки RESTful API к своему проекту с помощью фреймворка Django REST framework. Вы научитесь создавать сериализаторы для моделей и конкретно-прикладные представления API. Вы будете оперировать аутентификацией по API и реализуете разрешения для представлений API. Научитесь разрабатывать наборы представлений API и маршрутизаторы. В этой главе также будет рассказано о методах использования API с помощью библиотеки requests.

Глава 16 «*Разработка сервера чатов*» посящена технике применения приложения Django Channels с целью создания реально-временного чата-сервера для студентов. Вы научитесь реализовывать функциональности, которые опираются на асинхронную связь путем обмена данными по протоколу

WebSocket. Вы создадите WebSocket-потребителя с помощью Python и реализуете WebSocket-клиента с помощью JavaScript. Вы будете использовать хранилище Redis для настройки канального слоя и научитесь делать своего WebSocket-потребителя полностью асинхронным.

Глава 17 «*Выход в прямой эфир*» покажет, как создавать настроечные параметры для нескольких сред и устанавливать производственную среду на основе базы данных PostgreSQL, хранилища Redis, сервера приложения uWSGI, веб-сервера NGINX и асинхронного веб-сервера Daphne с помощью инструмента Docker Compose. Вы научитесь безопасно раздавать свой проект по протоколу HTTPS и использовать встроенный в Django фреймворк проверки системы. В этой главе также будет рассказано о методах разработки конкретно-прикладных промежуточных программных компонентов и конкретно-прикладных команд управления.

Что нужно, чтобы извлечь максимум пользы из этой книги

- Читатель должен обладать хорошими практическими знаниями Python.
- Читатель должен хорошо разбираться в HTML и JavaScript.
- Рекомендуется, чтобы читатель ознакомился с частями 1–3 практического руководства в официальной документации Django по адресу https://docs.djangoproject.com/en/4.1/intro/tutorial01/.

Используемые обозначения

В этой книге используется ряд текстовых обозначений.

`ИсходныйКодВТексте`: указывает слова исходного кода в тексте, имена таблиц базы данных, имена папок, имена файлов, расширения файлов, имена путей, фиктивные URL-адреса, вводимые пользователем данные и дескрипторы Twitter. Например, «Отредактировать файл `models.py` приложения `shop`».

Блок исходного кода задается, как показано ниже:

```
from django.contrib import admin
from .models import Post

admin.site.register(Post)
```

Когда мы хотим привлечь ваше внимание к определенной части блока исходного кода, соответствующие строки или элементы выделяются жирным шрифтом:

```
INSTALLED_APPS = [
    'django.contrib.admin',
```

```
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog.apps.BlogConfig',
]
```

Любые данные на входе или на выходе из команды командой оболочки записываются, как показано ниже:

```
python manage.py runserver
```

Жирный шрифт: выделяет новый термин, важное слово или слова, которые вы видите на экране. Например, слова в меню или диалоговых окнах пишутся в тексте следующим образом: «заполнить форму и нажать кнопку **Сохранить**».

Предупреждения или важные примечания помечаются так.

Советы и программные трюки выглядят так.

1

Разработка приложения для ведения блога

В этой книге вы научитесь разрабатывать профессиональные проекты Django. В данной главе будет продемонстрировано, как разрабатывать приложение Django, используя главные компоненты веб-фреймворка. Если Django у вас еще не установлен, то в первой части главы вы научитесь это делать.

Прежде чем приступать к первому проекту Django, давайте воспользуемся моментом, чтобы посмотреть, чему вы научитесь. В данной главе вы получите общий обзор веб-фреймворка. В ней вы ознакомитесь с различными самыми главными компонентами, служащими для создания полнофункционального веб-приложения: моделями, шаблонами, представлениями и URL-адресами. После ее прочтения вы будете иметь хорошее понимание того, как работает Django и как взаимодействуют различные компоненты веб-фреймворка.

В ней вы узнаете разницу между проектами и приложениями Django, а также ознакомитесь с наиболее важными настроечными параметрами Django. Вы разработаете простое приложение для ведения блога, которое позволит пользователям перемещаться по всем опубликованным постам и читать одиночные посты. Вы также создадите простой интерфейс администрирования, чтобы управлять постами и их публиковать. В следующих двух главах вы расширите приложение для ведения блога более продвинутыми функциональностями.

Данная глава должна послужить руководством по разработке полноценного приложения Django и дать представление о механизмах работы веб-фреймворка. Не беспокойтесь, если вы не понимаете всех аспектов веб-фреймворка. Различные компоненты веб-фреймворка будут подробно рассматриваться на протяжении всей этой книги.

В данной главе будут освещены следующие темы:

- установка Python;
- создание виртуальной среды Python;
- установка Django;

- создание и конфигурирование проекта Django;
- разработка приложения Django;
- конструирование моделей данных;
- создание и применение миграций моделей;
- создание сайта администрирования для моделей;
- работа с наборами запросов `QuerySet` и модельными менеджерами;
- формирование представлений, шаблонов и URL-адресов;
- понимание цикла «запрос/ответ» Django.

Установка языка Python

Django 4.1 поддерживает язык Python версий 3.8, 3.9 и 3.10. В приведенных в этой книге примерах мы будем использовать Python 3.10.6.

Если вы применяете Linux или macOS, то у вас, вероятно, Python уже установлен. Если же вы используете Windows, то на странице https://www.python.org/downloads/windows/ можно скачать установщик Python.

Откройте оболочку командной строки своего компьютера. Если вы используете macOS, то откройте каталог `/Applications/Utilities` в файловом менеджере **Finder**, затем дважды кликните по **Terminal**. Если вы используете Windows, то откройте меню **Пуск** и в поле поиска наберите `cmd`. Затем кликните по приложению **командной строки**, чтобы его открыть.

Проверьте, что на вашем компьютере Python установлен, набрав следующую ниже команду в командной оболочке:

```
python
```

Если вы видите что-то вроде следующего ниже, то Python на вашем компьютере установлен:

```
Python 3.10.6 (v3.10.6:9c7b4bd164, Aug  1 2022, 17:13:48) [Clang 13.0.0
(clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
```

Если установленная версия Python ниже 3.10 или если Python на вашем компьютере не установлен, то скачайте Python 3.10.6 на странице https://www.python.org/downloads/ и следуйте инструкциям по его установке. На странице скачиваний находится установщик Python для Windows, macOS и Linux.

На протяжении всей этой книги, когда в командной оболочке упоминается Python, мы будем использовать команду `python`, хотя в некоторых системах, возможно, потребуется применение команды `python3`. Если вы работаете на Linux или macOS, а в вашей системе используется Python 2, то для того чтобы задействовать установленную вами версию Python 3, вам нужно будет использовать команду `python3`.

В Windows команда `python` представляет исполняемый файл установки Python, которая используется по умолчанию, тогда как команда `py` – программу

быстрого запуска языка Python. Программа быстрого запуска языка Python для Windows была впервые представлена в Python версии 3.3. Она выясняет версии языка Python, которые были установлены на вашем компьютере, и автоматически переключается на последнюю версию. Если вы работаете с Windows, то рекомендуется заменить команду `python` командой `py`. Подробнее о программе быстрого запуска языка Python в Windows можно почитать на странице https://docs.python.org/3/using/windows.html#launcher.

Создание виртуальной среды Python

При написании приложений на Python обычно используются пакеты и модули, которые не включены в стандартную библиотеку Python. При этом некоторым приложениям Python может требоваться другая версия одного и того же модуля. Однако в масштабах всей системы можно устанавливать только определенную версию модуля. Если обновить версию модуля приложения, то это может привести к нарушению работы других приложений, которым требуется более старая версия этого модуля.

В целях решения указанной проблемы используются виртуальные среды Python. С помощью виртуальных сред модули Python можно устанавливать в изолированном месте, не устанавливая их глобально. Каждая виртуальная среда имеет свой собственный двоичный файл Python и свой собственный независимый набор пакетов Python, расположенных в каталоге `site-packages`.

Начиная с версии 3.3 Python идет в комплекте с библиотекой `venv`, которая обеспечивает поддержку создания облегченных виртуальных сред. Применяя модуль Python `venv` для создания изолированных сред Python, можно использовать разные версии пакетов для разных проектов. Еще одним преимуществом работы с `venv` является то, что для установки пакетов Python не понадобятся какие-либо административные привилегии.

Если вы работаете с Linux или macOS, то следующей ниже командой создайте изолированную среду:

```
python -m venv my_env
```

В случае если ваша система идет в комплекте с Python 2 и вы установили Python 3, то не забудьте применить команду `python3` вместо `python`.

Если вы используете Windows, то вместо этого примените следующую ниже команду:

```
py -m venv my_env
```

При этом будет использоваться программа быстрого запуска Python в Windows.

Приведенная выше команда создаст среду Python в новом каталоге с именем `my_env/`. Любые библиотеки Python, которые устанавливаются вами, пока

ваша виртуальная среда является активной, будут помещаться в каталог `my_env/lib/python3.10/site-packages`.

Если вы используете Linux или macOS, то выполните следующую ниже команду, чтобы активировать свою виртуальную среду:

```
source my_env/bin/activate
```

Если вы используете Windows, то вместо этого привлеките следующую ниже команду:

```
.\my_env\Scripts\activate
```

Приглашение командной оболочки будет содержать имя активной виртуальной среды, заключенное в круглые скобки, как показано ниже:

```
(my_env) zenx@pc:~ zenx$
```

Свою среду можно деактивировать в любое время с помощью команды `deactivate`. Более подробная информация о `venv` находится на странице https://docs.python.org/3/library/venv.htm.

Установка веб-фреймворка Django

Если вы уже установили Django 4.1, то этот раздел можно пропустить и перейти непосредственно к разделу «Создание первого проекта».

Django поставляется в виде модуля Python, и, следовательно, его можно устанавливать в любой среде Python. Если вы еще не установили Django, то ниже приведено краткое руководство по его установке на компьютер.

Установка Django с помощью pip

Система управления пакетами `pip` является предпочтительным методом установки Django. Python 3.10 идет в комплекте с предустановленной `pip`, однако инструкция по установке `pip` находится на странице https://pip.pypa.io/en/stable/installing/.

Выполните следующую ниже команду в командной оболочке, чтобы установить Django с помощью `pip`:

```
pip install Django~=4.1.0
```

Она установит последнюю версию Django 4.1 в каталог Python `site-packages/` вашей виртуальной среды.

Теперь мы проверим успешность установки Django. Выполните следующую ниже команду в командной оболочке:

```
python -m django --version
```

Если вы получите результат `4.1.X`, то, значит, Django был успешно установлен на вашем компьютере. Если вы получаете сообщение `No module named Django`, то, значит, Django на вашем компьютере не установлен. Если у вас возникли проблемы с установкой Django, то на странице https://docs.djangoproject.com/en/4.1/intro/install/ можно уточнить различные опции установки.

Веб-фреймворк Django можно устанавливать различными способами. С вариантами опций установки можно ознакомиться на странице https://docs.djangoproject.com/en/4.1/topics/install/.

Все используемые в этой главе пакеты Python включены в файл `requirements.txt` в исходном коде к данной главе. Следуйте инструкциям по установке каждого пакета Python в последующих разделах либо установите требующиеся пакеты сразу с помощью команды `pip install -r requirements.txt`.

Новые функциональные возможности Django 4

Django 4 вводит набор новых функциональных возможностей, включающий несколько обратно несовместимых изменений, в то же время объявляя о скорой замене других функциональностей и устраняя старые. Поскольку релиз Django 4 основан на времени, в нем нет кардинальных изменений, и приложения Django 3 легко мигрируются в версию 4.1. В отличие от релиза Django 3, куда впервые была включена поддержка интерфейса шлюза асинхронного сервера **ASGI**[1], в Django 4.0 добавлено несколько характерных особенностей, таких как функциональные ограничения по уникальности для моделей Django, встроенная поддержка кеширования данных с использованием сервера хранилища Redis, новая стандартная реализация часового пояса с применением стандартного пакета Python `zoneinfo`, новый механизм хеширования паролей `scrypt`, шаблонно-ориентированная прорисовка форм, а также другие новые второстепенные функциональности. В Django 4.0 отменяется поддержка Python 3.6 и 3.7. В нем также прекращается поддержка PostgreSQL 9.6, Oracle 12.2 и Oracle 18c. В Django 4.1 вводятся асинхронные обработчики представлений на основе классов, асинхронный интерфейс ORM-преобразователя, новая валидация модельных ограничений и новые шаблоны прорисовки форм. В версии 4.1 прекращается поддержка PostgreSQL 10 и MariaDB 10.2.

С полным списком изменений можно ознакомиться в примечаниях к релизу Django 4.0 на странице https://docs.djangoproject.com/en/dev/releases/4.0/ и примечаниях к релизу Django 4.1 на странице https://docs.djangoproject.com/en/4.1/releases/4.1/.

---

[1] Англ. Asynchronous Server Gateway Interface. – *Прим. перев.*

Общий обзор веб-фреймворка Django

Django – это веб-фреймворк, состоящий из набора компонентов, которые решают распространенные задачи веб-разработки. Компоненты Django слабо сцеплены между собой, и поэтому ими можно управлять независимо, что помогает разделять обязанности разных слоев веб-фреймворка; слой базы данных ничего не знает о том, как данные отображаются на странице, система шаблонов ничего не знает о веб-запросах и т. д.

Django обеспечивает максимальную возможность реиспользования исходного кода, следуя принципу DRY[1]. Django также способствует быстрой разработке и позволяет использовать меньше исходного кода, применяя динамические возможности языка Python, такие как интроспекция.

Подробнее о философии дизайна Django можно почитать на странице https://docs.djangoproject.com/en/4.1/misc/design-philosophies/.

Главные компоненты веб-фреймворка

Django подчиняется шаблону архитектурного дизайна **MTV** (**Model-Template-View**)[2]. Он немного похож на хорошо известный шаблон архитектурного дизайна **MVC** (**Model-View-Controller**)[3], где Template (Шаблон)[4] действует как View (Представление), а сам веб-фреймворк действует как Controller (Контроллер).

Обязанности в шаблоне архитектурного дизайна MTV Django распределены следующим образом:

- **модель** – определяет логическую структуру данных и является обработчиком данных между базой данных и их представлением;
- **шаблон** – это слой представления. В Django используется система текстовых шаблонов, в которой хранится все, что браузер прорисовывает на страницах;
- **представление** – взаимодействует с базой данных через модель и передает данные в шаблон для их прорисовки и просмотра.

Сам веб-фреймворк выступает в качестве контроллера. Он отправляет запрос в надлежащее представление в соответствии с конфигурацией URL-адреса.

При разработке любого проекта Django вы всегда будете работать с моделями, представлениями, шаблонами и URL-адресами. В этой главе вы узнаете, как они сочетаются друг с другом.

---

1 От англ. don't repeat yourself (не повторяйся). – *Прим. перев.*

2 Модель–шаблон–представление. – *Прим. перев.*

3 Модель–представление–контроллер. – *Прим. перев.*

4 Синоним «трафарет». В рамках Django «шаблон» (или трафарет) представляет собой некую заготовку страницы или пустой бланк, в котором выделено несколько пустых полей различной формы, служащий для размножения документов. Шаблон накладывается на контекстные данные и прорисовывается. См. https://ru.wikipedia.org/wiki/Трафарет. – *Прим. перев.*

Архитектура Django

На рис. 1.1 показано, как Django обрабатывает запросы и как различные главные компоненты Django – модели, шаблоны, URL-адреса и представления – управляют циклом запроса/ответа.

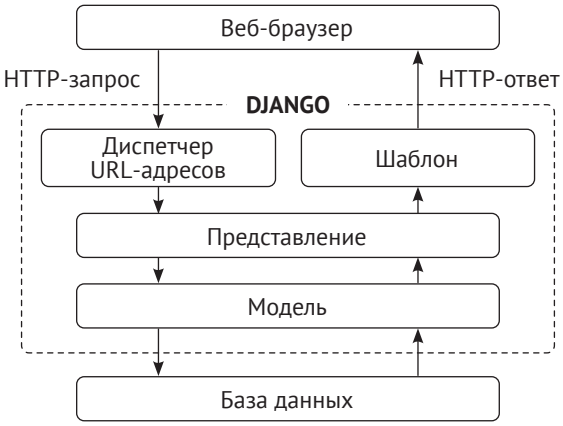

Рис. 1.1. Архитектура Django

Вот как Django оперирует HTTP-запросами и генерирует ответы:

1. Веб-браузер запрашивает страницу по ее URL-адресу, и веб-сервер передает HTTP-запрос веб-фреймворку Django.
2. Django просматривает свои сконфигурированные шаблоны URL-адресов и останавливается на первом, который совпадает с запрошенным URL-адресом.
3. Django исполняет представление, соответствующее совпавшему шаблону URL-адреса.
4. Представление потенциально использует модели данных, чтобы извлекать информацию из базы данных.
5. Модели данных обеспечивают определение данных и их поведение. Они используются для запроса к базе данных.
6. Представление прорисовывает[1] шаблон (обычно с использованием HTML), чтобы отображать данные на странице, и возвращает их вместе с HTTP-ответом.

[1] Согласно документации Django под рендерингом понимается взятие промежуточного представления шаблона вместе с контекстом и его превращение в итоговый поток байтов, который может быть передан клиенту (например, браузеру). На техническом языке это *интерполяция* (т. е. заполнение) шаблона контекстными данными и возврат результирующего строкового литерала, или *трансляция* данных в другой формат. В переводе при изложении тем, связанных с архитектурой MVP и шаблонами, используется термин «прорисовка» шаблона, а при обсуждени темы API – термин «трансляция» (см. https://docs.djangoproject.com/en/stable/ref/template-response/#the-rendering-process). – *Прим. перев.*

Мы вернемся к *циклу запроса/ответа* Django в конце этой главы в разделе «Цикл запроса/ответа».

В составе Django также имеются перехватчики[1], вызываемые внутри процесса запроса/ответа, которые называются промежуточными программными компонентами[2]. Промежуточные программные компоненты были намеренно исключены из этой диаграммы ради простоты. Вы будете использовать промежуточные программные компоненты в различных примерах этой книги, а о том, как создавать конкретно-прикладной промежуточный программный компонент, вы узнаете в главе 17 «Выход в прямой эфир».

Создание первого проекта

Ваш первый проект Django будет состоять из приложения для ведения блога. Мы начнем с создания проекта Django и приложения Django для ведения блога. Затем создадим модели данных и синхронизируем их с базой данных.

Django предоставляет команду, которая позволяет создавать изначальную файловую структуру проекта. Выполните следующую ниже команду в командной оболочке:

```
django-admin startproject mysite
```

Она создаст проект Django с именем `mysite`.

Во избежание конфликтов имен следует избегать именования проектов по имени встроенных модулей Python или Django.

Давайте взглянем на сгенерированную структуру проекта:

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

Внешний каталог `mysite/` является контейнером проекта. Он содержит следующие ниже файлы:

---

[1] Син. зацепки, хуки. – *Прим. перев.*

[2] Англ. middleware. – *Прим. перев.*

- `manage.py`: это утилита командной строки, используемая для взаимодействия с проектом. Редактировать этот файл не требуется;
- `mysite/`: это пакет проекта на языке Python; пакет состоит из следующих ниже файлов:
  - `__init__.py`: пустой файл, который сообщает Python, что каталог `mysite` нужно трактовать как модуль Python;
  - `asgi.py`: конфигурация для выполнения проекта в качестве приложения, работающего по протоколу интерфейса шлюза асинхронного сервера (**ASGI**) с ASGI-совместимыми веб-серверами. ASGI – это новый стандарт Python для асинхронных веб-серверов и приложений;
  - `settings.py`: здесь указаны настроечные параметры и конфигурация проекта и содержатся изначальные параметры со значениями, используемыми по умолчанию;
  - `urls.py`: место, где располагаются ваши шаблоны URL-адресов. Каждый URL-адрес, который определен здесь, соотносится с представлением;
  - `wsgi.py`: конфигурация для выполнения проекта в качестве приложения, работающего по протоколу интерфейса шлюза веб-сервера (**WSGI**)[1] с WSGI-совместимыми веб-серверами.

Применение первоначальных миграций базы данных

Для того чтобы хранить данные, приложениям Django требуется база данных. Упомянутый выше файл `settings.py` содержит конфигурацию базы данных проекта в настроечном параметре `DATABASES`. Изначально конфигурацией предусматривается использование базы данных SQLite3, если не указана иная. SQLite идет в комплекте с Python 3 и может применяться в любом приложении Python. SQLite – это облегченная база данных, которую можно использовать с Django для разработки. Если вы планируете развернуть свое приложение в производственной среде, то вам следует использовать полнофункциональную базу данных, такую как PostgreSQL, MySQL или Oracle. Более подробная информация о совместной работе базы данных с Django содержится по адресу https://docs.djangoproject.com/en/4.1/topics/install/#database-installation.

Файл `settings.py` также содержит настроечный параметр `INSTALLED_APPS` со списком, содержащим распространенные приложения Django, которые добавляются в ваш проект по умолчанию. Мы рассмотрим эти приложения позже в разделе «Настроечные параметры проекта».

Приложения Django содержат модели данных, которые соотносятся с таблицами базы данных. В разделе «Создание моделей данных блога» вы создадите свои собственные модели. Для того чтобы завершить настройку проекта, необходимо создать таблицы, ассоциированные с моделями стандартных

[1] Англ. Web Server Gateway Interface. – *Прим. перев.*

приложений Django, включенных в состав параметра `INSTALLED_APPS`. Django поставляется вместе с системой, которая помогает управлять миграциями баз данных.

Откройте приглашение командной оболочки и выполните следующие ниже команды:

```
cd mysite
python manage.py migrate
```

Вы увидите результат, который заканчивается следующими ниже строками:

```
Applying contenttypes.0001_initial... OK
Applying auth.0001_initial... OK
Applying admin.0001_initial... OK
Applying admin.0002_logentry_remove_auto_add... OK
Applying admin.0003_logentry_add_action_flag_choices... OK
Applying contenttypes.0002_remove_content_type_name... OK
Applying auth.0002_alter_permission_name_max_length... OK
Applying auth.0003_alter_user_email_max_length... OK
Applying auth.0004_alter_user_username_opts... OK
Applying auth.0005_alter_user_last_login_null... OK
Applying auth.0006_require_contenttypes_0002... OK
Applying auth.0007_alter_validators_add_error_messages... OK
Applying auth.0008_alter_user_username_max_length... OK
Applying auth.0009_alter_user_last_name_max_length... OK
Applying auth.0010_alter_group_name_max_length... OK
Applying auth.0011_update_proxy_permissions... OK
Applying auth.0012_alter_user_first_name_max_length... OK
Applying sessions.0001_initial... OK
```

Показанные выше строки – это применяемые веб-фреймворком Django миграции базы данных. В результате применения изначальных настроек в базе данных создаются таблицы для приложений, перечисленных в настроечном параметре `INSTALLED_APPS`.

Подробнее о команде управления `migrate` можно узнать в разделе «Создание и применение миграций» данной главы.

Запуск и выполнение сервера разработки

Django идет в комплекте вместе с облегченным веб-сервером с целью быстрого выполнения вашего исходного кода без необходимости тратить время на настройку производственного сервера. Во время работы сервера разработки он непрерывно проверяет наличие изменений в исходном коде. Он автоматически перезагружается, освобождая от необходимости перезагружать его вручную после изменения кода. Однако есть случаи, когда он может не

замечать некоторые действия, такие как добавление новых файлов в проект, поэтому в подобных случаях приходится перезапускать сервер вручную.

Запустите сервер разработки, набрав следующую ниже команду в командной оболочке:

```
python manage.py runserver
```

Вы должны увидеть что-то вроде этого:

```
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
January 01, 2022 - 10:00:00
Django version 4.0, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Теперь пройдите по URL-адресу `http://127.0.0.1:8000/` в своем браузере. Вы должны увидеть страницу, на которой указано, что проект успешно выполняется, как показано на рис. 1.2.

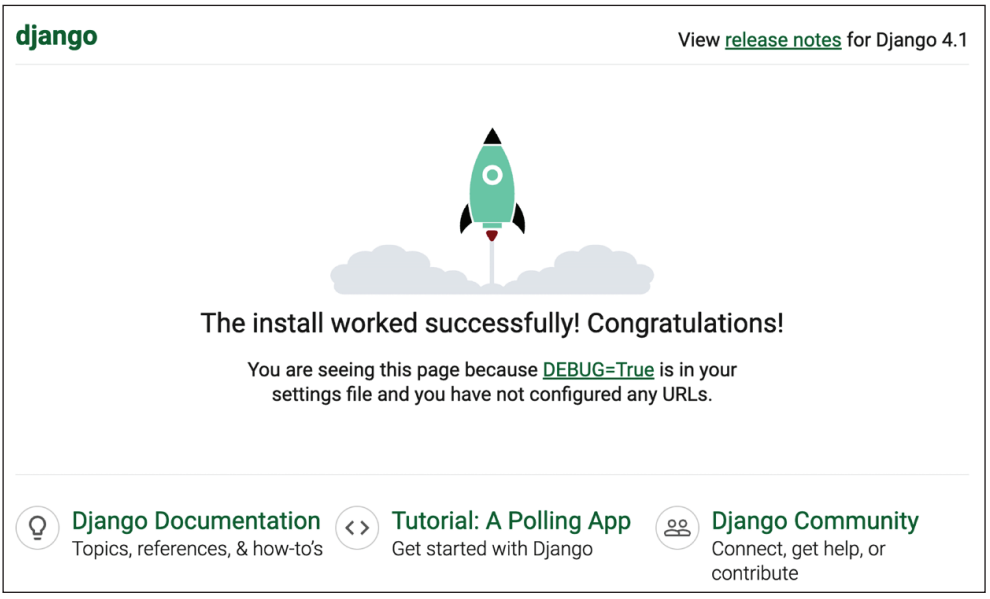

Рис. 1.2. Домашняя страница сервера разработки

Приведенный выше снимок экрана показывает, что Django работает. Если вы взглянете на свою консоль, то увидите выполняемый браузером запрос `GET`:

```
[01/Jan/2022 17:20:30] "GET / HTTP/1.1" 200 16351
```

Каждый HTTP-запрос регистрируется в консоли сервером разработки. Любая ошибка, возникающая во время работы сервера разработки, также будет появляться в консоли.

Сервер разработки можно выполнять на конкретно-прикладном хосте и порту либо сообщать Django, что нужно загружать определенный настроечный файл, как показано ниже:

```
python manage.py runserver 127.0.0.1:8001 --settings=mysite.settings
```

Когда приходится иметь дело с несколькими средами, требующими разных конфигураций, то следует создавать настроечный файл отдельно для каждой среды.

Этот сервер предназначен только для разработки и не подходит для производственного использования. Для того чтобы развернуть Django в производственной среде, необходимо его выполнять как приложение на основе WSGI с использованием такого веб-сервера, как Apache, Gunicorn или uWSGI, или же как приложение на основе ASGI с использованием такого сервера, как Daphne или Uvicorn. Более подробная информация о том, как разворачивать Django с различными веб-серверами, находится на странице https://docs.djangoproject.com/en/4.1/howto/deployment/wsgi/.

В главе 17 «Выход в прямой эфир» объясняется техника настраивания производственной среды под проекты Django.

Настроечные параметры проекта

Давайте откроем файл `settings.py` и взглянем на конфигурацию проекта. Несколько настроечных параметров уже внесены в указанный файл веб-фреймворком Django, но это лишь часть всех имеющихся параметров. Все настроечные параметры и их значения, которые используются по умолчанию, можно увидеть на странице https://docs.djangoproject.com/en/4.1/ref/settings/.

Давайте рассмотрим некоторые настроечные параметры проекта.

- `DEBUG` – это булев параметр, который включает и выключает режим отладки проекта. Если его значение установлено равным `True`, то Django будет отображать подробные страницы ошибок в случаях, когда приложение выдает неперехваченное исключение. При переходе в производственную среду следует помнить о том, что необходимо устанавливать его значение равным `False`. Никогда не развертывайте свой сайт в производственной среде с включенной отладкой, поскольку вы предоставите конфиденциальные данные, связанные с проектом.
- `ALLOWED_HOSTS` не применяется при включенном режиме отладки или при выполнении тестов. При перенесении своего сайта в производственную среду и установке параметра `DEBUG` равным `False` в этот настроечный

параметр следует добавлять свои домен/хост, чтобы разрешить ему раздавать ваш сайт Django.

- `INSTALLED_APPS` – это параметр, который придется редактировать во всех проектах. Он сообщает Django о приложениях, которые для этого сайта являются активными. По умолчанию Django вставляет следующие ниже приложения:
  - `django.contrib.admin`: сайт администрирования;
  - `django.contrib.auth`: фреймворк аутентификации;
  - `django.contrib.contenttypes`: фреймворк типов контента;
  - `django.contrib.sessions`: фреймворк сеансов;
  - `django.contrib.messages`: фреймворк сообщений;
  - `django.contrib.staticfiles`: фреймворк управления статическими файлами.
- `MIDDLEWARE` – подлежащие исполнению промежуточные программные компоненты.
- `ROOT_URLCONF` указывает модуль Python, в котором определены шаблоны корневых URL-адресов приложения.
- `DATABASES` – словарь, содержащий настроечные параметры всех баз данных, которые будут использоваться в проекте. Всегда должна существовать база данных, которая будет использоваться по умолчанию. В стандартной конфигурации используется база данных SQLite3, если не указана иная.
- `LANGUAGE_CODE` определяет заранее заданный языковой код этого сайта Django.
- `USE_TZ` сообщает Django, что нужно активировать/деактивировать поддержку часовых поясов. Django поставляется вместе с поддержкой дат и времен с учетом часовых поясов. Этот настроечный параметр получает значение `True` при создании нового проекта с помощью команды управления `startproject`.

Не волнуйтесь, если вы многое не понимаете из того, что здесь видите. Подробнее о различных настроечных параметрах Django можно узнать в последующих главах.

Проекты и приложения

На протяжении всей этой книги вы снова и снова будете сталкиваться с терминами «**проект**» и «**приложение**». В Django проектом считается установленный веб-фреймворк Django с несколькими настроечными параметрами. Приложение – это группа моделей, шаблонов, URL-адресов и представлений. Приложения взаимодействуют с веб-фреймворком с целью обеспечения определенных функциональностей, и их можно реиспользовать в разных проектах. Проект можно трактовать как свой собственный веб-сайт, содержащий несколько приложений, таких как блог, вики или форум, который другие проекты Django тоже могут использовать.

На рис. 1.3 показана структура проекта Django.

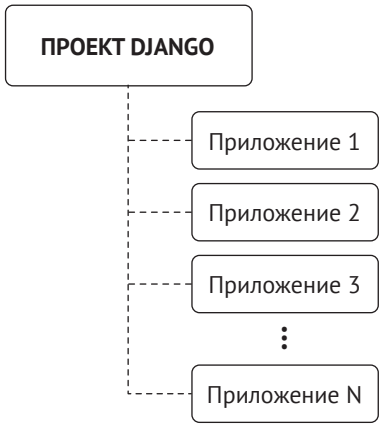

Рис. 1.3. Структура проекта/приложений Django

Создание приложения

Давайте создадим первое приложение Django. Мы разработаем приложение для ведения блога с нуля.

Выполните следующую ниже команду в командной оболочке из корневого каталога проекта:

```
python manage.py startapp blog
```

Она создаст базовую структуру приложения, которая будет выглядеть следующим образом:

```
blog/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

Ниже приведено описание этих файлов:

- `__init__.py`: пустой файл, который сообщает Python, что каталог `blog` нужно трактовать как модуль Python;
- `admin.py`: здесь вы регистрируете модели, чтобы включать их в состав сайта администрирования – этот сайт используется опционально, по вашему выбору;
- `apps.py`: содержит главную конфигурацию приложения `blog`;
- `migrations`: этот каталог будет содержать миграции базы данных приложения. Миграции позволяют Django отслеживать изменения модели

и соответствующим образом синхронизировать базу данных. Указанный каталог содержит пустой файл `__init__.py`;
- `models.py`: содержит относимые к приложению модели данных; все приложения Django должны иметь файл `models.py`, но его можно оставлять пустым;
- `tests.py`: здесь можно добавлять относимые к приложению тесты;
- `views.py`: здесь расположена логика приложения; каждое представление получает HTTP-запрос, обрабатывает его и возвращает ответ.

Когда структура приложения готова, можно приступать к разработке моделей данных блога.

Создание моделей данных блога

Напомним, что объект Python – это набор данных и методов. Класс – это концептуальная схема, которая объединяет данные и функциональности в единое целое. Создание нового класса влечет новый тип объекта, позволяя формировать экземпляры этого типа.

Модель Django – это источник информации и поведения данных. Она состоит из класса Python, который является подклассом `django.db.models.Model`. Каждой модели ставится в соответствие одна таблица базы данных, где каждый атрибут класса соотносится с полем базы данных. Когда вы будете создавать модель, Django будет предоставлять практичный API, чтобы легко запрашивать объекты в базе данных.

Сначала мы определим модели баз данных для приложения `blog`. Затем сгенерируем для этих моделей миграции базы данных, чтобы создать соответствующие таблицы базы данных. При применении миграций Django будет создавать таблицу по каждой модели, определенной в файле `models.py` приложения.

Создание модели поста

Сначала мы определим модель `Post`, которая позволит хранить посты блога в базе данных.

Добавьте следующие ниже строки в файл `models.py` приложения `blog`. Новые строки выделены жирным шрифтом:

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=250)
    slug = models.SlugField(max_length=250)
    body = models.TextField()
```

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

Это модель данных для постов блога. Посты будут иметь заголовок, короткую метку под названием `slug` и тело поста. Давайте взглянем на поля указанной модели:

- `title`: поле заголовка поста. Это поле с типом `CharField`, которое транслируется в столбец `VARCHAR` в базе данных SQL;
- `slug`: поле `SlugField`, которое транслируется в столбец `VARCHAR` в базе данных SQL. Слаг – это короткая метка, содержащая только буквы, цифры, знаки подчеркивания или дефисы. Пост с заголовком «*Django Reinhardt: A legend of Jazz*» мог бы содержать такой заголовок: «*django-reinhardt-legend-jazz*». В главе 2 «Усовершенствование блога за счет продвинутых функциональностей» мы будем использовать поле `slug` для формирования красивых и дружественных для поисковой оптимизации URL-адресов постов блога;
- `body`: поле для хранения тела поста. Это поле с типом `TextField`, которое транслируется в столбец `Text` в базе данных SQL.

В модельный класс также добавлен метод `__str__()`. Это метод Python, который применяется по умолчанию и возвращает строковый литерал с удобочитаемым представлением объекта. Django будет использовать этот метод для отображения имени объекта во многих местах, таких как его сайт администрирования.

Если вы использовали Python 2.x, то обратите внимание, что в Python 3 все строковые литералы изначально считаются кодированными в Юникоде; поэтому мы используем только метод `__str__()`. Метод `__unicode__()` из Python 2.x устарел.

Давайте посмотрим, как модель и ее поля будут транслированы в таблицу и столбцы базы данных. На следующей ниже диаграмме показана модель `Post` и соответствующая таблица базы данных, которую Django создаст, когда мы синхронизируем модель с базой данных.

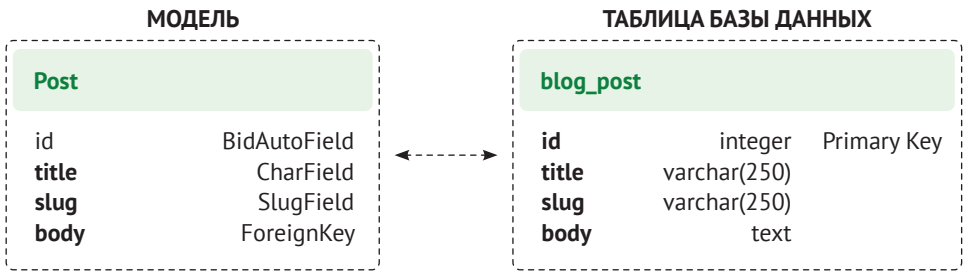

Рис. 1.4. Соответствие изначальной модели Post и таблицы базы данных

Django создаст столбец базы данных для каждого поля модели: `title`, `slug` и `body`. На рисунке хорошо видно, как каждый тип поля соответствует типу данных в базе данных.

Django по умолчанию добавляет поле автоматически увеличивающегося первичного ключа в каждую модель. Тип этого поля указывается в конфигурации каждого приложения либо глобально в настроечном параметре `DEFAULT_AUTO_FIELD`. При создании приложения командой `startapp` значение параметра `DEFAULT_AUTO_FIELD` по умолчанию имеет тип `BigAutoField`. Это 64-битное целое число, которое увеличивается автоматически в соответствии с доступными идентификаторами. Если не указывать первичный ключ своей модели, то Django будет добавлять это поле автоматически. В качестве первичного ключа можно также определить одно из полей модели, установив для него параметр `primary_key=True`.

Мы расширим модель `Post` дополнительными полями и поведением. После завершения мы синхронизируем ее с базой данных, создав миграцию в базе данных и применив ее.

Добавление полей даты/времени

Мы продолжим, добавив в модель `Post` различные поля даты/времени. Каждый пост будет публиковаться в определенную дату и время. Следовательно, необходимо иметь поле для хранения даты и времени публикации. Мы также хотим хранить дату и время создания объекта `Post` и его последнего изменения.

Отредактируйте файл `models.py` приложения `blog`, придав ему следующий вид. Новые строки выделены жирным шрифтом:

```
from django.db import models
from django.utils import timezone

class Post(models.Model):
    title = models.CharField(max_length=250)
    slug = models.SlugField(max_length=250)
    body = models.TextField()
    publish = models.DateTimeField(default=timezone.now)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)

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

В модель `Post` были добавлены следующие ниже поля:

- `publish`: поле с типом `DateTimeField`, которое транслируется в столбец `DATETIME` в базе данных SQL. Оно будет использоваться для хранения даты и времени публикации поста. По умолчанию значения поля за-

даются методом Django `timezone.now`. Обратите внимание, что для того, чтобы использовать этот метод, был импортирован модуль `timezone`. Метод `timezone.now` возвращает текущую дату/время в формате, зависящем от часового пояса. Его можно трактовать как версию стандартного метода Python `datetime.now` с учетом часового пояса;

- `created`: поле с типом `DateTimeField`. Оно будет использоваться для хранения даты и времени создания поста. При применении параметра `auto_now_add` дата будет сохраняться автоматически во время создания объекта;
- `updated`: поле с типом `DateTimeField`. Оно будет использоваться для хранения последней даты и времени обновления поста. При применении параметра `auto_now` дата будет обновляться автоматически во время сохранения объекта.

Определение предустановленного порядка сортировки

Посты блога обычно отображаются на странице в обратном хронологическом порядке (от самых новых к самым старым). В нашей модели мы определим заранее заданный порядок. Он будет применяться при извлечении объектов из базы данных, в случае если в запросе порядок не будет указан.

Отредактируйте файл `models.py` приложения `blog`, придав ему следующий вид. Новые строки выделены жирным шрифтом:

```
from django.db import models
from django.utils import timezone

class Post(models.Model):
    title = models.CharField(max_length=250)
    slug = models.SlugField(max_length=250)
    body = models.TextField()
    publish = models.DateTimeField(default=timezone.now)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)

    class Meta:
        ordering = ['-publish']

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

Внутрь модели был добавлен `Meta`-класс. Этот класс определяет метаданные модели. Мы используем атрибут `ordering`, сообщающий Django, что он должен сортировать результаты по полю `publish`. Указанный порядок будет применяться по умолчанию для запросов к базе данных, когда в запросе не

указан конкретный порядок. Убывающий порядок задается с помощью дефиса перед именем поля: `-publish`. По умолчанию посты будут возвращаться в обратном хронологическом порядке.

Добавление индекса базы данных

Давайте определим индекс базы данных по полю `publish`. Индекс повысит производительность запросов, фильтрующих или упорядочивающих результаты по указанному полю. Мы ожидаем, что многие запросы извлекут преимущества из этого индекса, поскольку для упорядочивания результатов мы по умолчанию используем поле `publish`.

Отредактируйте файл `models.py` приложения `blog`, придав ему следующий вид. Новые строки выделены жирным шрифтом:

```
from django.db import models
from django.utils import timezone

class Post(models.Model):
    title = models.CharField(max_length=250)
    slug = models.SlugField(max_length=250)
    body = models.TextField()
    publish = models.DateTimeField(default=timezone.now)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)

    class Meta:
        ordering = ['-publish']
        indexes = [
            models.Index(fields=['-publish']),
        ]

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

В `Meta`-класс модели была добавлена опция `indexes`. Указанная опция позволяет определять в модели индексы базы данных, которые могут содержать одно или несколько полей в возрастающем либо убывающем порядке, или функциональные выражения и функции базы данных. Был добавлен индекс по полю `publish`, а перед именем поля применен дефис, чтобы определить индекс в убывающем порядке. Создание этого индекса будет вставляться в миграции базы данных, которую мы сгенерируем позже для моделей блога.

Индексное упорядочивание в MySQL не поддерживается. Если в качестве базы данных вы используете MySQL, то убывающий индекс будет создаваться как обычный индекс.