

Владимир Дронов

Django 2.1

Практика

создания

веб-сайтов

на Python

Санкт-Петербург

«БХВ-Петербург»

2019

УДК 004.738.5+004.438Python
ББК 32.973.26-018.1
Д75

Дронов В. А.

Д75 Django 2.1. Практика создания веб-сайтов на Python. — СПб.: БХВ-Петербург, 2019. — 672 с.: ил. — (Профессиональное программирование) ISBN 978-5-9775-4058-2

Книга посвящена разработке веб-сайтов на Python с использованием веб-фреймворка Django 2.1. Рассмотрены основные функциональные возможности, необходимые для программирования сайтов общего назначения: модели, контроллеры, шаблоны, средства обработки пользовательского ввода, выгрузка файлов, разграничение доступа и др.

Рассказано о вспомогательных инструментах: посредниках, сигналах, средствах отправки электронной почты, подсистеме кэширования и пр. Описано форматирование текста посредством BBCode, обработка CAPTCHA, вывод графических миниатюр, аутентификация через социальные сети, интеграция с Bootstrap. Рассмотрено программирование веб-служб REST, использование административного веб-сайта Django, тестового сайта на Angular. Дан пример разработки полнофункционального веб-сайта — электронной доски объявлений. Исходный код доступен для загрузки с сайта издательства.

Для веб-программистов

УДК 004.738.5+004.438Python
ББК 32.973.26-018.1

Руководитель проекта	<i>Евгений Рыбаков</i>
Зав. редакцией	<i>Екатерина Сависте</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Дизайн серии	<i>Марины Дамбиевой</i>
Оформление обложки	<i>Карины Соловьевой</i>

Подписано в печать 28.02.19.
Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 54, 18.
Тираж 1000 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.
Отпечатано с готового оригинал-макета
ООО "Принт-М", 142300, М.О., г. Чехов, ул. Полиграфистов, д. 1

ISBN 978-5-9775-4058-2

© ООО "БХВ", 2019
© Оформление. ООО "БХВ-Петербург", 2019

Оглавление

Введение	17
Веб-фреймворк Django	17
Использованные программные продукты	19
Типографские соглашения	19
ЧАСТЬ I. ВВОДНЫЙ КУРС	21
Глава 1. Основные понятия Django. Вывод данных	23
1.1. Установка фреймворка	23
1.2. Проект Django	24
1.3. Отладочный веб-сервер Django	25
1.4. Приложения	27
1.5. Контроллеры	28
1.6. Маршруты и маршрутизатор	30
1.7. Модели	33
1.8. Миграции	35
1.9. Консоль Django	37
1.10. Работа с моделями	38
1.11. Шаблоны	42
1.12. Рендеринг шаблонов. Сокращения	43
1.13. Административный веб-сайт Django	45
1.14. Параметры полей и моделей	50
1.15. Редактор модели	51
Глава 2. Связи. Ввод данных. Статические файлы	53
2.1. Связи между моделями	53
2.2. Строковое представление модели	55
2.3. URL-параметры и параметризованные запросы	57
2.4. Обратное разрешение интернет-адресов	61
2.5. Формы, связанные с моделями	62
2.6. Контроллеры-классы	62
2.7. Наследование шаблонов	66
2.8. Статические файлы	69

ЧАСТЬ II. БАЗОВЫЕ ИНСТРУМЕНТЫ DJANGO.....	73
Глава 3. Создание и настройка проекта	75
3.1. Подготовительные действия	75
3.2. Создание проекта Django	77
3.3. Настройки проекта	77
3.3.1. Основные настройки	77
3.3.2. Параметры баз данных	78
3.3.3. Список зарегистрированных приложений	80
3.3.4. Список зарегистрированных посредников	81
3.3.5. Языковые настройки	82
3.4. Создание, настройка и регистрация приложений	85
3.4.1. Создание приложений	85
3.4.2. Настройка приложений	86
3.4.3. Регистрация приложения в проекте	86
3.5. Отладочный веб-сервер Django	87
Глава 4. Модели: базовые инструменты	89
4.1. Введение в модели	89
4.2. Объявление моделей	90
4.3. Объявление полей модели	91
4.3.1. Параметры, поддерживаемые полями всех типов	91
4.3.2. Классы полей моделей	94
4.4. Создание связей между моделями	97
4.4.1. Связь «один-со-многими»	97
4.4.2. Связь «один-с-одним»	100
4.4.3. Связь «многие-со-многими»	101
4.5. Параметры самой модели	103
4.6. Интернет-адрес модели и его формирование	106
4.7. Методы модели	107
4.8. Валидация модели. Валидаторы	109
4.8.1. Стандартные валидаторы Django	109
4.8.2. Вывод собственных сообщений об ошибках	113
4.8.3. Написание своих валидаторов	114
4.8.4. Валидация модели	115
Глава 5. Миграции	117
5.1. Формирование миграций	117
5.2. Файлы миграций	118
5.3. Выполнение миграций	119
5.4. Слияние миграций	119
5.5. Вывод списка миграций	120
5.6. Отмена всех миграций	121
Глава 6. Запись данных	122
6.1. Правка записей	122
6.2. Создание записей	123
6.3. Некоторые замечания о методе <i>save()</i>	124
6.4. Удаление записей	125

6.5. Особенности обработки связанных записей.....	125
6.5.1. Особенности обработки связи «один-со-многими»	126
6.5.2. Особенности обработки связи «один-с-одним»	127
6.5.3. Особенности обработки связи «многие-со-многими»	128
6.6. Произвольное переупорядочивание записей.....	129
6.7. Массовая запись данных	130
6.8. Выполнение валидации модели.....	131

Глава 7. Выборка данных..... 133

7.1. Извлечение значений из полей записи	133
7.2. Доступ к связанным записям	134
7.3. Выборка записей.....	135
7.3.1. Выборка всех записей	135
7.3.2. Извлечение одной записи.....	136
7.3.3. Получение количества записей в наборе	137
7.3.4. Поиск записи.....	138
7.3.5. Фильтрация записей	139
7.3.6. Написание условий фильтрации.....	140
7.3.7. Фильтрация по значениям полей связанных записей.....	142
7.3.8. Сравнение со значениями других полей.....	144
7.3.9. Сложные условия фильтрации	144
7.3.10. Выборка уникальных записей	145
7.3.11. Выборка указанного количества записей	145
7.4. Сортировка записей.....	146
7.5. Агрегатные вычисления	147
7.5.1. Вычисления по всем записям модели.....	147
7.5.2. Вычисления по группам записей.....	148
7.5.3. Агрегатные функции	149
7.6. Вычисляемые поля	151
7.6.1. Простейшие вычисляемые поля	152
7.6.2. Функции СУБД	153
7.6.3. Условные выражения СУБД.....	157
7.6.4. Вложенные запросы	159
7.7. Объединение наборов записей	160
7.8. Извлечение значений только из заданных полей.....	161
7.9. Получение значения из полей со списком.....	163

Глава 8. Маршрутизация..... 164

8.1. Как работает маршрутизатор.....	164
8.2. Списки маршрутов уровня проекта и уровня приложения	165
8.3. Объявление маршрутов	166
8.4. Передача данных в контроллеры.....	167
8.5. Именованные маршруты.....	168
8.6. Пространства имен. Корневое приложение	169
8.7. Указание шаблонных путей в виде регулярных выражений.....	170

Глава 9. Контроллеры-функции

9.1. Введение в контроллеры-функции	171
9.2. Как пишутся контроллеры-функции	171
9.2.1. Контроллеры, выполняющие одну задачу.....	172
9.2.2. Контроллеры, выполняющие несколько задач	173

9.3. Формирование ответа.....	174
9.3.1. Низкоуровневые средства для формирования ответа.....	174
9.3.2. Формирование ответа на основе шаблона.....	175
9.3.3. Класс <i>TemplateResponse</i> : отложенный рендеринг шаблона.....	177
9.4. Получение сведений о запросе.....	178
9.5. Перенаправление.....	180
9.6. Формирование интернет-адресов путем обратного разрешения.....	181
9.7. Выдача сообщений об ошибках и обработка особых ситуаций.....	182
9.8. Специальные ответы.....	183
9.8.1. Поточковый ответ.....	183
9.8.2. Отправка файлов.....	184
9.8.3. Отправка данных в формате JSON.....	185
9.9. Сокращения Django.....	185
9.10. Дополнительные настройки контроллеров.....	187
Глава 10. Контроллеры-классы.....	188
10.1. Введение в контроллеры-классы.....	188
10.2. Базовые контроллеры-классы.....	189
10.2.1. Контроллер <i>View</i> : диспетчеризация по HTTP-методу.....	189
10.2.2. Примесь <i>ContextMixin</i> : создание контекста шаблона.....	190
10.2.3. Примесь <i>TemplateResponseMixin</i> : рендеринг шаблона.....	190
10.2.4. Контроллер <i>TemplateView</i> : все вместе.....	191
10.3. Классы, выводющие сведения о выбранной записи.....	191
10.3.1. Примесь <i>SingleObjectMixin</i> : извлечение записи из модели.....	192
10.3.2. Примесь <i>SingleObjectTemplateResponseMixin</i> : рендеринг шаблона на основе найденной записи.....	193
10.3.3. Контроллер <i>DetailView</i> : все вместе.....	194
10.4. Классы, выводющие наборы записей.....	195
10.4.1. Примесь <i>MultipleObjectMixin</i> : извлечение набора записей из модели.....	195
10.4.2. Примесь <i>MultipleObjectTemplateResponseMixin</i> : рендеринг шаблона на основе набора записей.....	198
10.4.3. Контроллер <i>ListView</i> : все вместе.....	198
10.5. Классы, работающие с формами.....	199
10.5.1. Классы для вывода и валидации форм.....	200
10.5.1.1. Примесь <i>FormMixin</i> : создание формы.....	200
10.5.1.2. Контроллер <i>ProcessFormView</i> : вывод и обработка формы.....	201
10.5.1.3. Контроллер-класс <i>FormView</i> : создание, вывод и обработка формы.....	201
10.5.2. Классы для работы с записями.....	203
10.5.2.1. Примесь <i>ModelFormMixin</i> : создание формы, связанной с моделью.....	203
10.5.2.2. Контроллер <i>CreateView</i> : создание новой записи.....	204
10.5.2.3. Контроллер <i>UpdateView</i> : исправление записи.....	205
10.5.2.4. Примесь <i>DeletionMixin</i> : удаление записи.....	206
10.5.2.5. Контроллер <i>DeleteView</i> : удаление записи с подтверждением.....	206
10.6. Классы для вывода хронологических списков.....	207
10.6.1. Вывод последних записей.....	207
10.6.1.1. Примесь <i>DateMixin</i> : фильтрация записей по дате.....	207
10.6.1.2. Контроллер <i>BaseDateListView</i> : базовый класс.....	208
10.6.1.3. Контроллер <i>ArchiveIndexView</i> : вывод последних записей.....	209

10.6.2. Вывод записей по годам.....	210
10.6.2.1. Примесь <i>YearMixin</i> : извлечение года	210
10.6.2.2. Контроллер <i>YearArchiveView</i> : вывод записей за год.....	210
10.6.3. Вывод записей по месяцам	211
10.6.3.1. Примесь <i>MonthMixin</i> : извлечение месяца	211
10.6.3.2. Контроллер <i>MonthArchiveView</i> : вывод записей за месяц.....	212
10.6.4. Вывод записей по неделям.....	212
10.6.4.1. Примесь <i>WeekMixin</i> : извлечение номера недели.....	212
10.6.4.2. Контроллер <i>WeekArchiveView</i> : вывод записей за неделю	213
10.6.5. Вывод записей по дням	214
10.6.5.1. Примесь <i>DayMixin</i> : извлечение заданного числа	214
10.6.5.2. Контроллер <i>DayArchiveView</i> : вывод записей за день.....	214
10.6.6. Контроллер <i>TodayArchiveView</i> : вывод записей за текущее число	215
10.6.7. Контроллер <i>DateDetailView</i> : вывод одной записи за указанное число	215
10.7. Контроллер <i>RedirectView</i> : перенаправление.....	216
10.8. Контроллеры-классы смешанной функциональности	217
Глава 11. Шаблоны и статические файлы: базовые инструменты.....	220
11.1. Настройки проекта, касающиеся шаблонов	220
11.2. Вывод данных. Директивы	223
11.3. Теги шаблонизатора	224
11.4. Фильтры.....	231
11.5. Наследование шаблонов.....	238
11.6. Обработка статических файлов	239
11.6.1. Настройка подсистемы статических файлов.....	240
11.6.2. Обслуживание статических файлов	241
11.6.3. Формирование интернет-адресов статических файлов	241
Глава 12. Пагинатор	243
12.1. Класс <i>Paginator</i> : сам пагинатор. Создание пагинатора.....	243
12.2. Класс <i>Page</i> : часть пагинатора. Вывод пагинатора.....	245
Глава 13. Формы, связанные с моделями	247
13.1. Создание форм, связанных с моделями.....	247
13.1.1. Создание форм посредством фабрики классов	247
13.1.2. Создание форм путем быстрого объявления.....	249
13.1.3. Создание форм путем полного объявления.....	250
13.1.3.1. Как выполняется полное объявление	250
13.1.3.2. Параметры, поддерживаемые всеми типами полей	252
13.1.3.3. Доступные классы полей форм.....	253
13.1.3.4. Классы полей форм, применяемые по умолчанию	257
13.1.4. Задание элементов управления.....	258
13.1.4.1. Классы элементов управления	258
13.1.4.2. Элементы управления, применяемые по умолчанию	261
13.2. Обработка форм.....	262
13.2.1. Добавление записи посредством формы	262
13.2.1.1. Создание формы для добавления записи	262
13.2.1.2. Повторное создание формы	263
13.2.1.3. Валидация данных, занесенных в форму	263

13.2.1.4. Сохранение данных, занесенных в форму	264
13.2.1.5. Доступ к данным, занесенным в форму	265
13.2.2. Правка записи посредством формы	265
13.2.3. Некоторые соображения касательно удаления записей	267
13.3. Вывод форм на экран	267
13.3.1. Быстрый вывод форм	267
13.3.2. Расширенный вывод форм	269
13.4. Валидация в формах	272
13.4.1. Валидация полей формы	272
13.4.1.1. Валидация с применением валидаторов	272
13.4.1.2. Валидация путем переопределения методов формы	272
13.4.2. Валидация формы	273
Глава 14. Наборы форм, связанные с моделями	274
14.1. Создание наборов форм, связанных с моделями	274
14.2. Обработка наборов форм, связанных с моделями	277
14.2.1. Создание набора форм, связанного с моделью	277
14.2.2. Повторное создание набора форм	278
14.2.3. Валидация и сохранение набора форм	278
14.2.4. Доступ к данным, занесенным в набор форм	279
14.2.5. Реализация переупорядочивания записей	280
14.3. Вывод наборов форм на экран	281
14.3.1. Быстрый вывод наборов форм	282
14.3.2. Расширенный вывод наборов форм	283
14.4. Валидация в наборах форм	284
14.5. Встроенные наборы форм	285
14.5.1. Создание встроенных наборов форм	285
14.5.2. Обработка встроенных наборов форм	285
Глава 15. Разграничение доступа: базовые инструменты	287
15.1. Как работает подсистема разграничения доступа	287
15.2. Подготовка подсистемы разграничения доступа	288
15.2.1. Настройка подсистемы разграничения доступа	288
15.2.2. Создание суперпользователя	289
15.2.3. Смена пароля пользователя	290
15.3. Работа со списками пользователей и групп	290
15.3.1. Список пользователей	290
15.3.2. Группы пользователей. Список групп	292
15.4. Аутентификация и служебные процедуры	293
15.4.1. Контроллер <i>LoginView</i> : вход на сайт	293
15.4.2. Контроллер <i>LogoutView</i> : выход с сайта	295
15.4.3. Контроллер <i>PasswordChangeView</i> : смена пароля	297
15.4.4. Контроллер <i>PasswordChangeDoneView</i> : уведомление об успешной смене пароля	297
15.4.5. Контроллер <i>PasswordResetView</i> : отправка письма для сброса пароля	298
15.4.6. Контроллер <i>PasswordResetDoneView</i> : уведомление об отправке письма для сброса пароля	300
15.4.7. Контроллер <i>PasswordResetConfirmView</i> : собственно сброс пароля	301
15.4.8. Контроллер <i>PasswordResetCompleteView</i> : уведомление об успешном сбросе пароля	302

15.5. Получение сведений о текущем пользователе	303
15.6. Авторизация	305
15.6.1. Авторизация в контроллерах	305
15.6.1.1. Императивный подход к авторизации	305
15.6.1.2. Декларативная авторизация в контроллерах-функциях	306
15.6.1.3. Декларативная авторизация в контроллерах-классах	308
15.6.2. Авторизация в шаблонах	310

ЧАСТЬ III. РАСШИРЕННЫЕ ИНСТРУМЕНТЫ И ДОПОЛНИТЕЛЬНЫЕ БИБЛИОТЕКИ..... 311

Глава 16. Модели: расширенные инструменты.....	313
16.1. Управление выборкой полей	313
16.2. Связи «многие-со-многими» с дополнительными данными	317
16.3. Полиморфные связи	319
16.4. Наследование моделей	323
16.4.1. Прямое наследование моделей	323
16.4.2. Абстрактные модели	325
16.4.3. Прокси-модели	326
16.5. Создание своих диспетчеров записей	327
16.5.1. Создание диспетчеров записей	327
16.5.2. Создание диспетчеров обратной связи	329
16.6. Создание своих наборов записей	330
16.7. Управление транзакциями	333
16.7.1. Всё или ничего: два высокоуровневых режима управления транзакциями	333
16.7.1.1. Ничего: режим по умолчанию	333
16.7.1.2. Всё: режим для максималистов	334
16.7.2. Управление транзакциями на низком уровне	335
16.7.2.1. Включение режима «всё» на уровне контроллера	335
16.7.2.2. Обработка подтверждения транзакции	336
16.7.2.3. Выключение режима «всё» для контроллера	336
16.7.2.4. Управление транзакциями вручную	336

Глава 17. Формы и наборы форм: расширенные инструменты и дополнительная библиотека.....	338
17.1. Формы, не связанные с моделями	338
17.2. Наборы форм, не связанные с моделями	339
17.3. Расширенные средства для вывода форм и наборов форм	341
17.3.1. Указание CSS-стилей для форм	341
17.3.2. Настройка выводимых форм	341
17.3.3. Настройка наборов форм	342
17.4. Библиотека Django Simple Captcha: поддержка CAPTCHA	343
17.4.1. Установка Django Simple Captcha	343
17.4.2. Использование Django Simple Captcha	344
17.4.3. Настройка Django Simple Captcha	345
17.4.4. Дополнительные команды <code>captcha_clean</code> и <code>captcha_create_pool</code>	346
17.5. Дополнительные настройки проекта, имеющие отношение к формам	346

Глава 18. Шаблоны: расширенные инструменты и дополнительные библиотеки	348
18.1. Библиотека <code>django-precise-bbcode</code> : поддержка BBCode	348
18.1.1. Установка <code>django-precise-bbcode</code>	349
18.1.2. Поддерживаемые BBCode-теги	349
18.1.3. Обработка BBCode	350
18.1.3.1. Обработка BBCode в процессе вывода	350
18.1.3.2. Хранение BBCode в модели	351
18.1.4. Создание дополнительных BBCode-тегов	352
18.1.5. Создание смайликов	354
18.1.6. Настройка <code>django-precise-bbcode</code>	355
18.2. Библиотека <code>django-bootstrap4</code> : интеграция с Bootstrap	356
18.2.1. Установка <code>django-bootstrap4</code>	356
18.2.2. Использование <code>django-bootstrap4</code>	357
18.2.3. Настройка <code>django-bootstrap4</code>	362
18.3. Написание своих фильтров и тегов	364
18.3.1. Организация исходного кода	364
18.3.2. Написание фильтров	364
18.3.2.1. Написание и использование простейших фильтров	364
18.3.2.2. Управление заменой недопустимых знаков HTML	366
18.3.3. Написание тегов	367
18.3.3.1. Написание тегов, выводящих элементарные значения	368
18.3.3.2. Написание шаблонных тегов	369
18.3.4. Регистрация фильтров и тегов	370
18.4. Переопределение шаблонов	372
Глава 19. Обработка выгруженных файлов	374
19.1. Подготовка подсистемы обработки выгруженных файлов	374
19.1.1. Настройка подсистемы обработки выгруженных файлов	374
19.1.2. Указание маршрута для выгруженных файлов	376
19.2. Хранение файлов в моделях	376
19.2.1. Типы полей модели, предназначенные для хранения файлов	377
19.2.2. Поля, валидаторы и элементы управления форм, служащие для указания файлов	379
19.2.3. Обработка выгруженных файлов	380
19.2.4. Вывод выгруженных файлов	382
19.2.5. Удаление выгруженного файла	383
19.3. Хранение путей к файлам в моделях	383
19.4. Низкоуровневые средства для сохранения выгруженных файлов	384
19.4.1. Класс <code>UploadedFile</code> : выгруженный файл. Сохранение выгруженных файлов	384
19.4.2. Вывод выгруженных файлов низкоуровневыми средствами	386
19.5. Библиотека <code>django-cleanup</code> : автоматическое удаление ненужных файлов	387
19.6. Библиотека <code>easy-thumbnails</code> : вывод миниатюр	388
19.6.1. Установка <code>easy-thumbnails</code>	388
19.6.2. Настройка <code>easy-thumbnails</code>	389
19.6.2.1. Пресеты миниатюр	389
19.6.2.2. Остальные параметры библиотеки	391
19.6.3. Вывод миниатюр в шаблонах	393

19.6.4. Хранение миниатюр в моделях	394
19.6.5. Дополнительная команда <i>thumbnail_cleanup</i>	395
Глава 20. Разграничение доступа: расширенные инструменты и дополнительная библиотека	396
20.1. Настройки проекта, касающиеся разграничения доступа	396
20.2. Работа с пользователями	397
20.2.1. Создание пользователей	397
20.2.2. Работа с паролями	397
20.3. Аутентификация и выход с сайта	398
20.4. Валидация паролей	399
20.4.1. Стандартные валидаторы паролей	399
20.4.2. Написание своих валидаторов паролей	401
20.4.3. Выполнение валидации паролей	402
20.5. Библиотека Python Social Auth: регистрация и вход через социальные сети	403
20.5.1. Создание приложения «ВКонтакте»	403
20.5.2. Установка и настройка Python Social Auth	405
20.5.3. Использование Python Social Auth	406
20.6. Указание своей модели пользователя	406
20.7. Создание своих прав пользователя	408
Глава 21. Посредники и обработчики контекста	409
21.1. Посредники	409
21.1.1. Стандартные посредники	409
21.1.2. Порядок выполнения посредников	410
21.1.3. Написание своих посредников	411
21.1.3.1. Посредники-функции	411
21.1.3.2. Посредники-классы	412
21.2. Обработчики контекста	414
Глава 22. Cookie, сессии, всплывающие сообщения и подписывание данных	416
22.1. Cookie	416
22.2. Сессии	418
22.2.1. Настройка сессий	419
22.2.2. Использование сессий	421
22.2.3. Дополнительная команда <i>clearsessions</i>	423
22.3. Всплывающие сообщения	423
22.3.1. Настройка всплывающих сообщений	423
22.3.2. Уровни всплывающих сообщений	424
22.3.3. Создание всплывающих сообщений	425
22.3.4. Вывод всплывающих сообщений	426
22.3.5. Объявление своих уровней всплывающих сообщений	428
22.4. Подписывание данных	428
Глава 23. Сигналы	431
23.1. Обработка сигналов	431
23.2. Встроенные сигналы Django	433
23.3. Объявление своих сигналов	437

Глава 24. Отправка электронных писем	439
24.1. Настройка подсистемы отправки электронных писем	439
24.2. Низкоуровневые инструменты для отправки писем	441
24.2.1. Класс <i>EmailMessage</i> : обычное электронное письмо	441
24.2.2. Формирование писем на основе шаблонов	443
24.2.3. Использование соединений. Массовая рассылка писем	443
24.2.4. Класс <i>EmailMultiAlternatives</i> : электронное письмо, состоящее из нескольких частей	444
24.3. Высокоуровневые инструменты для отправки писем	445
24.3.1. Отправка писем по произвольным адресам	445
24.3.2. Отправка писем зарегистрированным пользователям	446
24.3.3. Отправка писем администраторам и редакторам сайта	447
Глава 25. Кэширование	449
25.1. Кэширование на стороне сервера	449
25.1.1. Подготовка подсистемы кэширования на стороне сервера	449
25.1.1.1. Настройка подсистемы кэширования на стороне сервера	450
25.1.1.2. Создание таблицы для хранения кэша	452
25.1.2. Высокоуровневые средства кэширования	452
25.1.2.1. Кэширование всего веб-сайта	453
25.1.2.2. Кэширование на уровне отдельных контроллеров	454
25.1.2.3. Управление кэшированием	455
25.1.3. Низкоуровневые средства кэширования	456
25.1.3.1. Кэширование фрагментов веб-страниц	456
25.1.3.2. Кэширование произвольных значений	458
25.2. Кэширование на стороне клиента	461
25.2.1. Автоматическая обработка заголовков	461
25.2.2. Условная обработка запросов	462
25.2.3. Прямое указание параметров кэширования	463
25.2.4. Запрет кэширования	464
Глава 26. Административный веб-сайт Django	465
26.1. Подготовка административного веб-сайта к работе	465
26.2. Регистрация моделей на административном веб-сайте	466
26.3. Редакторы моделей	467
26.3.1. Параметры списка записей	467
26.3.1.1. Параметры списка записей: состав выводимого списка	467
26.3.1.2. Параметры списка записей: фильтрация и сортировка	471
26.3.1.3. Параметры списка записей: прочие	475
26.3.2. Параметры страниц добавления и правки записей	476
26.3.2.1. Параметры страниц добавления и правки записей: набор выводимых полей	476
26.3.2.2. Параметры страниц добавления и правки записей: элементы управления	480
26.3.2.3. Параметры страниц добавления и правки записей: прочие	482
26.3.3. Регистрация редакторов на административном веб-сайте	483
26.4. Встроенные редакторы	484
26.4.1. Объявление встроенного редактора	484

26.4.2. Параметры встроенного редактора	485
26.4.3. Регистрация встроенного редактора	487
26.5. Действия	487

Глава 27. Разработка веб-служб REST. Библиотека Django REST

framework.....	490
27.1. Установка и подготовка к работе Django REST framework	491
27.2. Введение в Django REST framework. Вывод данных.....	492
27.2.1. Сериализаторы	492
27.2.2. Веб-представление JSON	494
27.2.3. Вывод данных на стороне клиента.....	496
27.2.4. Первый принцип REST: идентификация ресурса по интернет-адресу	498
27.3. Ввод и правка данных	501
27.3.1. Второй принцип REST: идентификация действия по HTTP-методу	501
27.3.2. Парсеры веб-форм	506
27.4. Контроллеры-классы Django REST framework	507
27.4.1. Контроллер-класс низкого уровня	507
27.4.2. Контроллеры-классы высокого уровня: комбинированные и простые	508
27.5. Метаконтроллеры	509
27.6. Разграничение доступа.....	511
27.6.1. Третий принцип REST: данные клиента хранятся на стороне клиента	511
27.6.2. Классы разграничения доступа	512

Глава 28. Средства диагностики и отладки.....

28.1. Средства диагностики	514
28.1.1. Настройка средств диагностики	514
28.1.2. Объект сообщения	515
28.1.3. Форматировщики.....	516
28.1.4. Фильтры	517
28.1.5. Обработчики	518
28.1.6. Регистраторы.....	523
28.1.7. Пример настройки диагностических средств.....	525
28.2. Средства отладки.....	527
28.2.1. Веб-страница сообщения об ошибке	527
28.2.2. Отключение кэширования статических файлов.....	529

Глава 29. Публикация готового веб-сайта.....

29.1. Подготовка веб-сайта к публикации	531
29.1.1. Веб-страницы с сообщениями об ошибках и их шаблоны.....	531
29.1.2. Указание настроек эксплуатационного режима.....	533
29.1.3. Подготовка статических файлов	534
29.1.4. Удаление ненужных данных.....	536
29.1.5. Окончательная проверка веб-сайта	536
29.2. Публикация веб-сайта с использованием веб-сервера Apache	537
29.2.1. Подготовка платформы для публикации	537
29.2.2. Конфигурирование веб-сайта	539
29.2.3. Особенности публикации веб-сайта, работающего по протоколу HTTPS	541

ЧАСТЬ IV. ПРАКТИЧЕСКОЕ ЗАНЯТИЕ: РАЗРАБОТКА ВЕБ-САЙТА.....	543
Глава 30. Дизайн. Вспомогательные веб-страницы.....	545
30.1. План веб-сайта	545
30.2. Подготовка проекта и приложения <i>main</i>	546
30.2.1. Создание и настройка проекта.....	546
30.2.2. Создание и настройка приложения <i>main</i>	547
30.3. Базовый шаблон.....	547
30.4. Главная веб-страница	553
30.5. Вспомогательные веб-страницы.....	556
Глава 31. Работа с пользователями и разграничение доступа.....	558
31.1. Модель пользователя.....	558
31.2. Основные веб-страницы: входа, профиля и выхода	560
31.2.1. Веб-страница входа	560
31.2.2. Веб-страница пользовательского профиля.....	562
31.2.3. Веб-страница выхода.....	564
31.3. Веб-страницы правки личных данных пользователя.....	565
31.3.1. Веб-страница правки основных сведений	565
31.3.2. Веб-страница правки пароля.....	568
31.4. Веб-страницы регистрации и активации пользователей	569
31.4.1. Веб-страницы регистрации нового пользователя	569
31.4.1.1. Форма для занесения сведений о новом пользователе	569
31.4.1.2. Средства для регистрации пользователя.....	571
31.4.1.3. Средства для отправки писем с требованиями активации	573
31.4.2. Веб-страницы активации пользователя	575
31.5. Веб-страница удаления пользователя	577
31.6. Инструменты для администрирования пользователей.....	579
Глава 32. Рубрики	582
32.1. Модели рубрик.....	582
32.1.1. Базовая модель рубрик.....	582
32.1.2. Модель надрубрик	583
32.1.3. Модель подрубрик.....	584
32.2. Инструменты для администрирования рубрик	585
32.3. Вывод списка рубрик в панели навигации	587
Глава 33. Объявления	590
33.1. Подготовка к обработке выгруженных файлов.....	590
33.2. Модели объявлений и дополнительных иллюстраций	591
33.2.1. Модель самих объявлений	592
33.2.2. Модель дополнительных иллюстраций	594
33.2.3. Реализация удаления объявлений в модели пользователя	594
33.3. Инструменты для администрирования объявлений.....	595
33.4. Вывод объявлений	596
33.4.1. Вывод списка объявлений.....	596
33.4.1.1. Форма поиска и контроллер списка объявлений.....	596
33.4.1.2. Реализация корректного возврата.....	598
33.4.1.3. Шаблон страницы списка объявлений	599

33.4.2. Вывод сведений о выбранном объявлении.....	602
33.4.3. Вывод последних 10 объявлений на главной веб-странице.....	606
33.5. Работа с объявлениями.....	607
33.5.1. Вывод объявлений, оставленных текущим пользователем.....	607
33.5.2. Добавление, правка и удаление объявлений	608
Глава 34. Комментарии.....	612
34.1. Подготовка к выводу CAPTCHA.....	612
34.2. Модель комментария.....	613
34.3. Вывод и добавление комментариев	614
34.4. Отправка уведомлений о появлении новых комментариев.....	617
Глава 35. Веб-служба REST	619
35.1. Веб-служба	619
35.1.1. Подготовка к разработке веб-службы	619
35.1.2. Список объявлений.....	620
35.1.3. Сведения о выбранном объявлении	621
35.1.4. Вывод и добавление комментариев	622
35.2. Тестовый клиентский веб-сайт	624
35.2.1. Подготовка к разработке тестового веб-сайта	624
35.2.2. Метамодули. Метамодуль приложения <i>AppModule</i> . Маршрутизация в Angular.....	626
35.2.3. Компоненты. Компонент приложения <i>AppComponent</i> . Стартовая веб-страница	631
35.2.4. Службы. Служба <i>BbService</i> . Внедрение зависимостей.....	633
35.2.5. Компонент списка объявлений <i>BbListComponent</i> . Директивы. Фильтры. Связывание данных	637
35.2.6. Компонент сведений об объявлении <i>BbDetailComponent</i> . Двустороннее связывание данных	640
Заключение.....	645
Приложение. Описание электронного архива.....	647
Предметный указатель	649

Введение

Иногда случается так, что какой-либо многообещающий программный продукт или программная технология с шумом и треском появляются на рынке, напропалую грозят всех конкурентов если и не полностью уничтожить, то отодвинуть в глубокую тень, привлекают к себе внимание всех интересующихся информационными технологиями, после чего тихо уходят с рынка, и о них никто более не вспоминает.

Так вот — все это не о Django. Появившись в 2005 году — именно тогда вышла его первая версия, — он остается одним из популярнейших программных инструментов, предназначенных для разработки веб-сайтов.

Веб-фреймворк Django

Язык программирования Python исключительно развит сам по себе, но основную мощь ему придают всевозможные дополнительные библиотеки, которых существует превеликое множество. Есть библиотеки для научных расчетов, систем машинного зрения, программирования игр, обычных «настольных» приложений и, разумеется, веб-сайтов.

Среди последних особняком стоит ряд библиотек, реализующих большую часть функциональности сайта. Эти библиотеки самостоятельно взаимодействуют с базами данных, обрабатывают клиентские запросы и формируют ответы, реализуют разграничение доступа, пуская к закрытым разделам сайта лишь тех посетителей, что перечислены в особом списке, и даже рассылают электронные письма. Разработчику остается только написать код, который генерирует веб-страницы сайта на основе данных, извлеченных из базы. И задача эта несравнимо менее трудоемкая, чем написание всей функциональности сайта, что называется, с нуля.

Такая всеобъемлющая библиотека напоминает готовый каркас (по-английски — *framework*), на который разработчик конкретного сайта «навешивает» свои узлы, механизмы и детали декора. Именно поэтому библиотеки подобного рода носят название *веб-фреймворков*, или просто *фреймворков*.

Один из фреймворков, написанных на языке Python, — Django. Среди всех такого рода разработок его стоит выделить особо. Хотя бы потому, что он, как было отме-

чено ранее, невероятно популярен. Более того, это наиболее часто применяемый на практике веб-фреймворк, разработанный на Python. И тому есть ряд причин.

- Django — это следование современным стандартам веб-разработки. В их числе: архитектура «модель-контроллер-шаблон», использование миграций для внесения изменений в базу данных и принцип «написанное однажды применяется везде» (или, другими словами, «не повторяйся»).
- Django — это полнофункциональный фреймворк. Для разработки среднестатистического сайта вам достаточно установить только его. Никаких дополнительных библиотек, необходимых, чтобы ваше веб-творение хотя бы заработало, ставить не придется.
- Django — это высокоуровневый фреймворк. Типовые задачи, наподобие соединения с базой данных, обработки данных, полученных от пользователя, сохранения выгруженных пользователем файлов, он выполняет самостоятельно. А еще он предоставляет полнофункциональную подсистему разграничения доступа и исключительно мощный и удобно настраиваемый административный веб-сайт, которые, в случае применения любого другого фреймворка, нам пришлось бы писать самостоятельно.
- Django — это удобство разработки. Легкий и быстрый отладочный веб-сервер, развитый механизм миграций, уже упомянутый административный веб-сайт — все это существенно упрощает программирование.
- Django — это дополнительные библиотеки. Нужен вывод графических миниатюр? Требуется обеспечить аутентификацию посредством социальных сетей? Необходима поддержка CAPTCHA? Для всего этого существуют библиотеки, которые нужно только установить.
- Django — это Python. Исключительно мощный и, вероятно, самый лаконичный язык из всех, что применяются в промышленном программировании.

Эта книга посвящена Django. Она описывает его наиболее важные и часто применяемые на практике функциональные возможности, ряд низкоуровневых инструментов, которые также могут пригодиться во многих случаях, и некоторые доступные для фреймворка дополнительные библиотеки. А в конце, в качестве практического упражнения, она описывает разработку полнофункционального сайта электронной доски объявлений.

ВНИМАНИЕ!

Автор предполагает, что читатели этой книги знакомы с языком разметки веб-страниц HTML, технологией каскадных таблиц стилей CSS, языком программирования веб-сценариев JavaScript и универсальным языком программирования Python. Описания всех этих технологий в книге приводиться не будут.

МАТЕРИАЛЫ ПОЛНОФУНКЦИОНАЛЬНОГО САЙТА

Электронный архив с исходным кодом сайта электронной доски объявлений можно скачать с FTP-сервера издательства «БХВ-Петербург» по ссылке <ftp://ftp.bhv.ru/9785977540582.zip> или со страницы книги на сайте www.bhv.ru (см. приложение).

Использованные программные продукты

Автор применял в работе следующие версии ПО:

- Microsoft Windows 10, русская 64-разрядная редакция со всеми установленными обновлениями;
- Python 3.6.5, 64-разрядная редакция;
- Django 2.1.3;
- Django Simple Captcha — 0.5.9;
- django-precise-bbcode — 1.2.9;
- django-bootstrap4 — 0.0.6 (разработка) и 0.0.7 (последующая проверка);
- Pillow — 5.2.0;
- django-cleanup — 2.1.0;
- easy-thumbnails — 2.5;
- Python Social Auth — 2.1.0;
- Django REST framework — 3.8.2;
- django-cors-headers — 2.4.0;
- mod-wsgi — 4.6.4;
- Angular — 6.1.7.

Типографские соглашения

В книге будут часто приводиться форматы написания различных языковых конструкций, применяемых в Python и Django. В них использованы особые типографские соглашения, которые мы сейчас изучим.

ВНИМАНИЕ!

Все эти типографские соглашения применяются автором только в форматах написания языковых конструкций Python и Django. В реальном программном коде они не имеют смысла.

- В угловые скобки (<>) заключаются наименования различных значений, которые дополнительно выделяются курсивом. В реальный код, разумеется, должны быть подставлены реальные значения. Например:

```
django-admin startproject <имя проекта>
```

Здесь вместо подстроки *имя проекта* должно быть подставлено реальное имя проекта.

- В квадратные скобки ([]) заключаются необязательные фрагменты кода. Например:

```
django-admin startproject <имя проекта> [<путь к папке проекта>]
```

Здесь *путь к папке проекта* может указываться, а может и не указываться.

- ❑ Слишком длинные, не помещающиеся на одной строке языковые конструкции автор разрывал на несколько строк и в местах разрывов ставил знаки ↵. Например:

```
background: url("bg.jpg") left / auto 100% no-repeat, ↵  
url("bg.jpg") right / auto 100% no-repeat;
```

Приведенный код разбит на две строки, но должен быть набран в одну. Символ ↵ при этом нужно удалить.

- ❑ Троекочием (. . .) помечены пропущенные ради сокращения объема текста фрагменты кода.

```
INSTALLED_APPS = [  
    . . .  
    'bboard.apps.BboardConfig',  
]
```

Здесь пропущены все элементы списка, присваиваемого переменной `INSTALLED_APPS`, кроме последнего.

Обычно такое можно встретить в исправленных впоследствии фрагментах кода — приведены лишь собственно исправленные выражения, а оставшиеся неизменными пропущены. Также троекочие используется, чтобы показать, в какое место должен быть вставлен вновь написанный код, — в начало исходного фрагмента, в его конец или в середину, между уже присутствующими в нем выражениями.

ЕЩЕ РАЗ ВНИМАНИЕ!

Все приведенные здесь типографские соглашения имеют смысл лишь в форматах написания конструкций Python и Django. В коде примеров используются только знак ↵ и троекочие.



ЧАСТЬ I

Вводный курс

Глава 1. Основные понятия Django. Вывод данных

Глава 2. Связи. Ввод данных. Статические файлы



ГЛАВА 1

Основные понятия Django. Вывод данных

Давайте сразу же приступим к делу. Прямо сейчас, в этой главе, мы установим сам фреймворк Django и начнем разработку простенького веб-сайта — электронной доски объявлений.

НА ЗАМЕТКУ

Эта книга не содержит описания языка программирования Python. Если вам, уважаемый читатель, необходима помощь в его освоении, обратитесь к другим учебным пособиям. Полное описание Python можно найти на его «домашнем» сайте <https://www.python.org/>, там же имеются дистрибутивы его исполняющей среды (интерпретатора) в различных редакциях для разных операционных систем.

1.1. Установка фреймворка

Начиная с версии Python 3.4, в составе исполняющей среды этого языка поставляется утилита `pip`, с помощью которой очень удобно выполнять установку любых дополнительных библиотек. Эта утилита самостоятельно ищет указанную при ее запуске библиотеку в штатном репозитории PyPI (Python Package Index, реестр пакетов Python) — интернет-хранилище самых разных библиотек для Python. Найдя запрошенную библиотеку, `pip` самостоятельно загружает и устанавливает наиболее подходящую ее редакцию, при этом загружая и устанавливая также и все библиотеки, которые она использует для работы.

Запустим командную строку и отдадим в ней команду на установку Django, которая вполне понятна безо всяких комментариев:

```
pip install django
```

ВНИМАНИЕ!

Если исполняющая среда Python установлена в папке *Program Files* или *Program Files (x86)*, для выполнения установки любых дополнительных библиотек командную строку следует запустить с повышенными правами. Для этого надо найти в меню **Пуск** пункт **Командная строка** (в зависимости от версии Windows он может находиться в группе **Стандартные** или **Служебные**), щелкнуть на нем правой кнопкой мыши и выбрать

в появившемся контекстном меню пункт **Запуск от имени администратора** (в Windows 10 этот пункт находится в подменю **Дополнительно**).

Помимо Django, эта команда установит также библиотеку `pytz`, выполняющую обработку значений даты и времени с учетом временных зон и используемую упомянутым ранее фреймворком в работе. Не удаляйте эту библиотеку!

Спустя некоторое время фреймворк будет установлен, о чем `pip` нам обязательно сообщит (приведены номера версий Django и `pytz`, актуальные на момент подготовки книги):

```
Successfully installed django-2.1.3 pytz-2018.7
```

Теперь мы можем начинать разработку нашего первого веб-сайта.

1.2. Проект Django

Следующее, что нам нужно сделать, — создать новый проект. *Проектом* называется совокупность всего программного кода, составляющего разрабатываемый сайт. Можно даже сказать, что проект — это и есть наш сайт. Физически он представляет собой папку, в которой находятся разнообразные файлы с исходным кодом и другие папки (назовем ее *папкой проекта*).

Давайте же создадим новый, пока еще пустой проект Django, которому дадим имя `samplesite`. Для этого в запущенной ранее командной строке перейдем в папку, в которой должна находиться папка проекта, и отдадим команду:

```
django-admin startproject samplesite
```

Утилита `django-admin` служит для выполнения разнообразных административных задач. В частности, команда `startproject` указывает ей создать новый проект с именем, записанным после этой команды.

В папке, в которую мы ранее перешли, будет создана следующая структура файлов и папок:

```
samplesite
  manage.py
  samplesite
    __init__.py
    settings.py
    urls.py
    wsgi.py
```

«Внешняя» папка `samplesite` — это, как нетрудно догадаться, и есть папка проекта. Как видим, ее имя совпадает с именем проекта, записанным в вызове утилиты `django-admin`. А содержимое этой папки таково:

- `manage.py` — программный файл с кодом одноименной утилиты, с использованием которой производятся различные действия над самим проектом. Впрочем, единственное, чем она занимается, — вызывает утилиту `django-admin`, передавая ей все полученные команды и конфигурируя ее для обработки текущего проекта;

□ «внутренняя» папка `samplesite` — формирует пакет языка Python, содержащий модули, которые относятся к проекту целиком и задают его конфигурацию (в частности, ключевые настройки). Название этого пакета совпадает с названием проекта и менять его не стоит — в противном случае придется вносить в код обширные правки.

В документации по Django этот пакет не имеет какого-либо ясного и однозначного названия. Поэтому, чтобы избежать путаницы, давайте назовем его *пакетом конфигурации*.

Пакет конфигурации включает в себя такие модули:

- `__init__.py` — пустой файл, сообщающий Python, что папка, в которой он находится, является полноценным пакетом;
- `settings.py` — модуль с настройками самого проекта. Включает описание конфигурации базы данных проекта, пути ключевых папок, важные параметры, связанные с безопасностью, и пр.;
- `urls.py` — модуль с маршрутами уровня проекта (о них мы поговорим позже);
- `wsgi.py` — модуль, связывающий проект с веб-сервером. Используется при публикации готового сайта в Интернете. Мы будем рассматривать этот модуль в *главе 29*.

Еще раз отметим, что пакет конфигурации хранит настройки, относящиеся к самому проекту и влияющие на все приложения, что входят в состав этого проекта (о приложениях мы поведем разговор очень скоро).

Проект Django мы можем поместить в любое место файловой системы компьютера. Мы также можем переименовать папку проекта. В результате всего этого проект не потеряет своей работоспособности.

1.3. Отладочный веб-сервер Django

В процессе разработки сайта нам придется неоднократно открывать его в веб-обозревателе для тестирования. Если бы мы использовали другую фундаментальную программную платформу, например PHP, и другой фреймворк, такой как Yii или Laravel, — нам пришлось бы устанавливать на свой компьютер программу веб-сервера. Но в случае с Django делать этого не нужно — в состав Django входит *отладочный веб-сервер*, написанный на самом языке Python, не требующий сложной настройки и всегда готовый к работе. Чтобы запустить его, следует в командной строке перейти непосредственно в папку проекта (именно в нее, а не в папку, в которой находится папка проекта!) и отдать команду:

```
manage.py runserver
```

Здесь мы пользуемся уже утилитой `manage.py`, сгенерированной программой `django-admin` при создании проекта. Команда `runserver`, которую мы записали после имени этой утилиты, как раз и запускает отладочный веб-сервер.

Последний сразу же выдаст нам небольшое сообщение, говорящее о том, что код сайта загружен, проверен на предмет ошибок и запущен в работу, и что сам сайт теперь доступен по интернет-адресу **http://127.0.0.1:8000/** (хотя намного удобнее набрать адрес **http://localhost:8000/** — он проще запоминается). Как видим, отладочный сервер по умолчанию работает через TCP-порт 8000 (впрочем, при необходимости можно использовать другой).

Запустим веб-обозреватель и наберем в нем один из интернет-адресов нашего сайта. Мы увидим информационную страничку, предоставленную самим Django и сообщающую, что сайт, хоть еще и «пуст», но, в целом, работает (рис. 1.1).

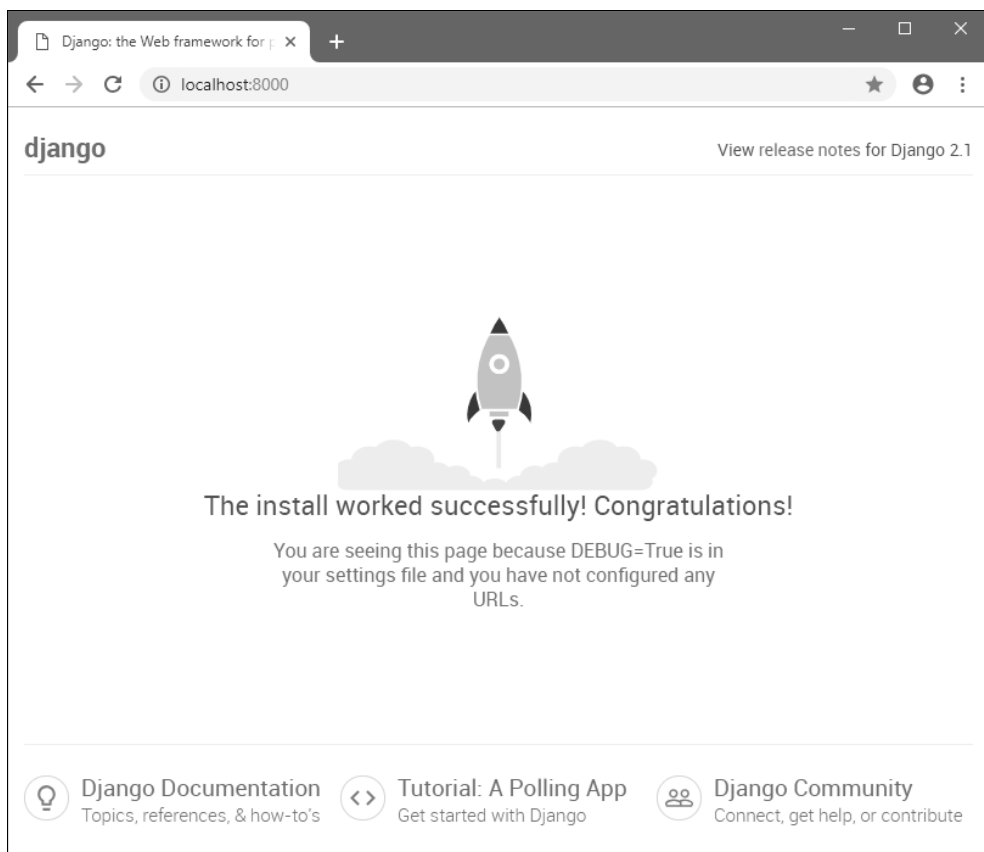


Рис. 1.1. Информационная веб-страница Django, сообщающая о работоспособности вновь созданного «пустого» сайта

Остановить отладочный веб-сервер можно, переключившись в экземпляре командной строки, в которой он был запущен, и нажав комбинацию клавиш `<Ctrl>+<Break>`.

1.4. Приложения

«Пустой» проект не содержит вообще никакой функциональности. (Вывод информационной страницы, которую мы только что наблюдали, не в счет.) Нам понадобится ее добавить. И реализуется эта функциональность в отдельных приложениях.

Приложение в терминологии Django — это отдельный фрагмент функциональности разрабатываемого сайта, более или менее независимый от других таких же фрагментов и входящий в состав проекта. Приложение может реализовывать работу целого сайта, его раздела или же какой-либо внутренней подсистемы сайта, используемой другими приложениями.

Любое приложение представляется обычным пакетом Python (*пакет приложения*), в котором находятся модули с программным кодом. Этот пакет находится в папке проекта — там же, где располагается пакет конфигурации. Имя пакета приложения станет именем самого приложения.

Нам нужно сделать так, чтобы наш сайт выводил перечень объявлений, оставленных посетителями. Для этого мы создадим новое приложение, которое незатейливо назовем `bboard`.

Новое приложение создается следующим образом. Сначала остановим отладочный веб-сервер. В командной строке проверим, находимся ли мы в папке проекта, и наберем команду:

```
manage.py startapp bboard
```

Команда `startapp` утилиты `manage.py` запускает создание нового «пустого» приложения, чье имя указано после этой команды.

Посмотрим, что же создала утилита `manage.py`. Прежде всего, это папка `bboard`, формирующая одноименный пакет приложения и расположенная в папке проекта. В ней находятся следующие папки и файлы:

- `migrations` — папка вложенного пакета, в котором будут сохраняться модули сгенерированных Django миграций (о них разговор обязательно пойдет, но позже). Пока что в папке находится лишь пустой файл `__init__.py`, помечающий ее как полноценный пакет Python;
- `__init__.py` — пустой файл, сигнализирующий языку Python, что эта папка — пакет;
- `admin.py` — модуль административных настроек и классов-редакторов;
- `apps.py` — модуль с настройками приложения;
- `models.py` — модуль с моделями;
- `tests.py` — модуль с тестирующими процедурами;
- `views.py` — модуль с контроллерами.

Пока что все это выглядит для нас как китайская грамота. Немного терпения — все это мы обязательно рассмотрим.

ВНИМАНИЕ!

Подсистема тестирования кода, реализованная в Django, в этой книге не рассматривается, поскольку автор не считает ее сколь-нибудь полезной.

Теперь давайте зарегистрируем только что созданное приложение в проекте. Найдем в пакете конфигурации файл `settings.py` (о котором уже упоминалось ранее), откроем его в текстовом редакторе и отыщем следующий фрагмент кода:

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

Список, хранящийся в переменной `INSTALLED_APPS`, перечисляет все приложения, зарегистрированные в проекте и участвующие в его работе. Все эти приложения поставляются в составе Django и реализуют работу какой-либо из встроенных подсистем фреймворка. Так, приложение `django.contrib.auth` реализует работу подсистемы разграничения доступа, а приложение `django.contrib.sessions` — подсистемы, обслуживающих серверные сессии.

В этой теплой компании явно не хватает нашего приложения `bboard`. Добавим его, включив в список новый элемент:

```
INSTALLED_APPS = [  
    . . .  
    'bboard.apps.BboardConfig',  
]
```

Обратим внимание на три важных момента. Во-первых, элемент списка приложений должен представлять собой строку с путем к классу `BboardConfig`, описывающему конфигурацию приложения и объявленному в упомянутом ранее модуле `apps.py`, что хранится в пакете приложения. Во-вторых, этот путь указывается в том формате, в котором записываются пути к модулям в стандарте языка Python (т. е. отдельные фрагменты пути разделяются точками, а не обратными слешами). В-третьих, этот путь указывается относительно папки проекта.

Сохраним и закроем файл `settings.py`. Но запускать отладочный веб-сервер пока не станем. Вместо этого сразу же напишем первый в нашей практике Django-программирования контроллер.

1.5. Контроллеры

Контроллер Django — это код, запускаемый в ответ на поступление клиентского запроса, который содержит интернет-адрес определенного формата. Именно в контроллерах выполняются все действия по подготовке данных для вывода, равно как и обработка данных, поступивших от посетителя.

ВНИМАНИЕ!

В документации по Django используется термин «view» (вид, или представление). Автор книги считает его неудачным и предпочитает применять термин «контроллер», тем более что это устоявшееся название программных модулей такого типа.

Контроллер Django может представлять собой как функцию (*контроллер-функция*), так и класс (*контроллер-класс*). Первые более универсальны, но зачастую трудоемки в программировании, вторые позволяют выполнить типовые задачи, наподобие вывода списка каких-либо позиций, минимумом кода. И первые, и вторые мы обязательно рассмотрим в последующих главах.

Для хранения кода контроллеров изначально предназначается модуль `views.py`, создаваемый в каждом пакете приложения. Однако ничто не мешает нам поместить контроллеры в другие модули, благо Django не предъявляет к организации кода контроллеров никаких специальных требований.

Давайте напишем контроллер, который будет выводить... нет, не список объявлений — этого списка у нас пока нет (у нас и базы данных-то, можно сказать, нет), а пока только текст, сообщающий, что будущие посетители сайта со временем увидят на этой страничке список объявлений. Это будет контроллер-функция.

Откроем модуль `views.py` пакета приложения `bboard`, удалим имеющийся там небольшой код и заменим его кодом из листинга 1.1.

Листинг 1.1. Простейший контроллер-функция, выводящий текстовое сообщение

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Здесь будет выведен список объявлений.")
```

Наш контроллер — это, собственно, функция `index()`. Единственное, что она делает, — отправляет клиенту текстовое сообщение: **Здесь будет выведен список объявлений**. Но это только пока...

Любой контроллер-функция в качестве единственного обязательного параметра принимает экземпляр класса `HttpRequest`, хранящий различные сведения о полученном запросе: запрашиваемый интернет-адрес, данные, полученные от посетителя, служебную информацию от самого веб-обозревателя и пр. По традиции этот параметр называется `request`. В нашем случае мы его никак не используем.

В теле функции мы создаем экземпляр класса `HttpResponse` (он объявлен в модуле `django.http`), который будет представлять отправляемый клиенту ответ. Содержимое этого ответа — собственно текстовое сообщение — мы указываем единственным параметром конструктора этого класса. Готовый экземпляр класса мы возвращаем из функции в качестве результата.

Что ж, теперь мы с гордостью можем считать себя программистами — поскольку уже самостоятельно написали какой-никакой программный код. Осталось запус-

тить отладочный веб-сервер, набрать в любимом веб-обозревателе адрес вида **http://localhost:8000/bboard/** и посмотреть, что получится...

Минуточку! А с чего мы взяли, что при наборе такого интернет-адреса Django запустит на выполнение именно написанный нами контроллер-функцию `index()`? Ведь мы нигде явно не связали интернет-адрес с контроллером. Но как это сделать?..

1.6. Маршруты и маршрутизатор

Сделать это очень просто. Нужно всего лишь:

- объявить связь интернет-адреса определенного формата (*шаблонного интернет-адреса*) с определенным контроллером — иначе говоря, *маршрут*.

Шаблонный интернет-адрес должен содержать только путь, без названия протокола, адреса хоста, номера порта, набора GET-параметров и имени якоря (поэтому его часто называют *шаблонным путем*). Он должен завершаться символом слеша (напротив, начальный слеш недопустим);

- оформить все объявленные нами маршруты в виде *списка маршрутов*;
- оформить маршруты в строго определенном формате, чтобы подсистема *маршрутизатора* смогла использовать готовый список в работе.

При поступлении любого запроса от клиента Django разбирает его на составные части (чем занимается целая группа программных модулей, называемых *посредниками* и описываемых в *главе 21*), извлекает запрошенный посетителем интернет-адрес, удаляет из него все составные части, за исключением пути, который передает маршрутизатору. Последний последовательно сравнивает его с шаблонными адресами, записанными в списке маршрутов. Как только будет найдено совпадение, маршрутизатор выясняет, какой контроллер связан с совпавшим шаблонным адресом, и передает этому контроллеру управление.

Давайте подумаем. Чтобы при наборе интернет-адреса **http://localhost:8000/bboard/** запускался только что написанный нами контроллер `index()`, нам нужно связать таковой с шаблонным адресом **bboard/**. Сделаем это.

В *разд. 1.2*, знакомясь с проектом, мы заметили хранящийся в пакете конфигурации модуль `urls.py`, в котором записываются маршруты уровня проекта. Давайте откроем этот модуль в текстовом редакторе и посмотрим, что он содержит (листинг 1.2).

Листинг 1.2. Изначальное содержимое модуля `urls.py` пакета конфигурации

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

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

Список маршрутов, оформленный в виде обычного списка Python, присваивается переменной `urlpatterns`. Каждый элемент списка маршрутов (т. е. каждый маршрут) должен представляться в виде результата, возвращаемого функцией `path()` из модуля `django.urls`. Последняя в качестве параметров принимает строку с шаблонным интернет-адресом и ссылкой на контроллер-функцию.

В качестве второго параметра функцией `path()` также может быть принят список маршрутов уровня приложения. Кстати, этот вариант демонстрируется в выражении, задающем единственный маршрут в листинге 1.2. Мы рассмотрим его потом.

А сейчас давайте добавим в список новый маршрут, связывающий шаблонный адрес **bboard/** и контроллер-функцию `index()`. Для чего дополним имеющийся в модуле `urls.py` код согласно листингу 1.3.

Листинг 1.3. Новое содержимое модуля `urls.py` пакета конфигурации

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

from bboard.views import index

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

Сохраним исправленный файл, запустим отладочный веб-сервер и наберем в веб-обозревателе интернет-адрес **http://localhost:8000/bboard/**. Мы увидим текстовое сообщение, сгенерированное нашим контроллером (рис. 1.2).

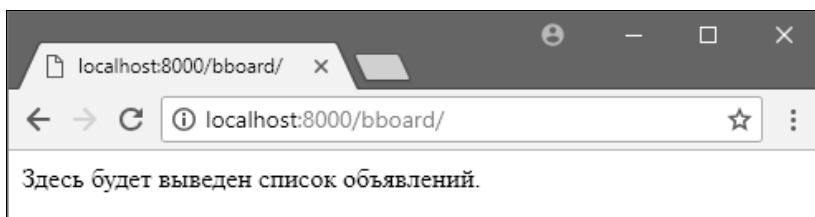


Рис. 1.2. Результат работы нашего первого контроллера — простое текстовое сообщение

Что ж, наши первые контроллер и маршрут работают, и по этому поводу можно порадоваться. Но лишь до поры до времени. Как только мы начнем создавать сложные сайты, состоящие из нескольких приложений, количество маршрутов в списке вырастет до таких размеров, что мы просто запутаемся в них. Поэтому создатели Django настоятельно рекомендуют применять для формирования списков маршрутов другой подход, о котором мы сейчас поговорим.

Маршрутизатор Django при просмотре списка маршрутов не требует, чтобы интернет-адрес, полученный из клиентского запроса, и шаблонный адрес, записанный

в очередном маршруте, совпадали полностью. Достаточно лишь того факта, что шаблонный адрес совпадает с началом реального. В таком случае шаблонизатор удаляет из реального адреса его начальную часть (префикс), совпавшую с шаблонным адресом, и запускает на исполнение указанный в маршруте контроллер.

Но, как было сказано ранее, функция `path()` позволяет указать во втором параметре вместо ссылки на контроллер-функцию другой список маршрутов. То есть мы можем указать для любого маршрута другой, *вложенный* в него список маршрутов. В таком случае маршрутизатор выполнит просмотр маршрутов, входящих в состав вложенного списка, используя для сравнения реальный интернет-адрес с уже удаленным из него префиксом.

Исходя из всего этого, мы можем создать иерархию списков маршрутов. В списке, созданном у самого проекта (*списке маршрутов уровня проекта*), мы укажем маршруты, которые указывают на вложенные списки маршрутов, записанные в отдельных приложениях проекта (*списки маршрутов уровня приложения*). А в последних мы уже запишем все контроллеры, что составляют программную логику нашего сайта.

Что ж, так и сделаем. И начнем со списка маршрутов уровня приложения `bboard`. Создадим в пакете этого приложения (т. е. в папке `bboard`) файл `urls.py` и занесем в него код из листинга 1.4.

Листинг 1.4. Код модуля `urls.py` пакета приложения `bboard`

```
from django.urls import path

from .views import index

urlpatterns = [
    path('', index),
]
```

Пустая строка, переданная первым параметром в функцию `path()`, обозначает корень пути из маршрута предыдущего уровня вложенности (*родительского*).

Наконец, исправим код модуля `urls.py` из пакета конфигурации, как показано в листинге 1.5.

Листинг 1.5. Окончательный код модуля `urls.py` пакета конфигурации

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

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


Вложенный список маршрутов, указываемый во втором параметре функции `path()`, должен представлять собой результат, возвращенный функцией `include()` из модуля `django.urls`. Единственным параметром эта функция принимает строку с путем к модулю, где записан список маршрутов.

Как только наш сайт получит запрос с интернет-адресом **http://localhost:8000/bboard/**, маршрутизатор обнаружит, что этот адрес совпадает с шаблонным адресом **bboard/**, записанным в первом маршруте из листинга 1.5. Он удалит из полученного в запросе адреса префикс, соответствующий шаблонному адресу, и получит пустую строку. Далее последует загрузка вложенного списка маршрутов из модуля `urls.py` пакета приложения `bboard`. Полученный интернет-адрес, представляющий собой пустую строку, совпадет с первым же маршрутом из вложенного списка, в результате чего запустится записанный в этом маршруте контроллер-функция `index()`, и на экране появится уже знакомое нам текстовое сообщение (см. рис. 1.2).

Поскольку зашла речь о вложенных списках маршрутов, давайте посмотрим на выражение, создающее второй маршрут из списка уровня проекта:

```
path('admin/', admin.site.urls),
```

Этот маршрут связывает шаблонный интернет-адрес **admin/** со списком маршрутов, возвращенным свойством `urls` экземпляра класса `AdminSite`, что хранится в переменной `site` и представляет текущий административный веб-сайт Django. Следовательно, набрав интернет-адрес **http://localhost:8000/admin/**, мы попадем на этот административный сайт (более подробно об административном сайте, встроеном во фреймворк, мы поговорим позже).

1.7. Модели

Настала пора сделать так, чтобы вместо намозолившего глаза текстового сообщения выводились реальные объявления, взятые из информационной базы. Если бы мы писали сайт на «чистом» Python, нам бы пришлось вручную создать в базе данных таблицу со всеми необходимыми полями и написать код, который будет открывать базу, считывать из нее данные и преобразовывать в нужный вид. Та еще работенка...

Однако мы работаем с Django — лучшим в мире веб-фреймворком. И для реализации хранения любых сущностей строго определенной структуры нам понадобится всего лишь объявить один-единственный класс, называемый *моделью*.

Модель — это однозначное и исчерпывающее описание сущности, хранящейся в базе данных в виде класса Python. Класс модели описывает таблицу базы данных, в которой будет храниться набор сущностей, и содержит атрибуты класса (в других языках программирования их называют свойствами класса, или статическими свойствами), каждый из которых описывает одно из полей этой таблицы. Можно сказать, что модель — это представление таблицы и ее полей средствами Python.

Отдельный экземпляр класса модели представляет отдельную конкретную сущность, извлеченную из базы, т. е. отдельную запись соответствующей таблицы. Пользуясь объявленными в модели атрибутами класса, мы можем получать значения, хранящиеся в полях записи, равно как и записывать в них новые значения.

Помимо этого, класс модели предоставляет инструменты для выборки сущностей из базы, их фильтрации и сортировки на основе заданных критериев. Полученный результат представляется в виде последовательности экземпляров класса модели.

Модели объявляются на уровне приложения. Объявляющий их код должен записываться в модуль `models.py` пакета приложения. Изначально этот модуль пуст.

Давайте объявим модель `Bb`, которая будет представлять объявление, со следующими полями:

- `title` — заголовок объявления, содержащий название продаваемого товара (тип — строковый, длина — 50 символов). Поле, обязательное к заполнению;
- `content` — сам текст объявления, описание товара (тип — `memo`);
- `price` — цена (тип — вещественное число);
- `published` — дата публикации (тип — дата и время, значение по умолчанию — текущие дата и время, индексированное).

Завершим работу отладочного веб-сервера. Откроем модуль `models.py` пакета приложения `bboard` и запишем в него код, объявляющий класс модели `Bb` (листинг 1.6).

Листинг 1.6. Код класса модели `Bb`

```
from django.db import models

class Bb(models.Model):
    title = models.CharField(max_length=50)
    content = models.TextField(null=True, blank=True)
    price = models.FloatField(null=True, blank=True)
    published = models.DateTimeField(auto_now_add=True, db_index=True)
```

Сама модель должна быть подклассом класса `Model` из модуля `django.db.models`. Отдельные поля модели, как говорилось ранее, оформляются как атрибуты класса, а в качестве значений им присваиваются экземпляры классов, представляющих поля разных типов и объявленных в том же модуле. Параметры полей указываются в конструкторах классов полей в виде значений именованных параметров.

Давайте рассмотрим использованные нами классы полей и их параметры:

- `CharField` — обычное строковое поле фиксированной длины. Допустимая длина значения указывается параметром `max_length` конструктора;
- `TextField` — текстовое поле неограниченной длины, или `memo`-поле. Присвоив параметрам `null` и `blank` конструктора значения `True`, мы укажем, что это поле можно не заполнять (по умолчанию любое поле обязательно к заполнению);

- `FloatField` — поле для хранения вещественных чисел. Оно также необязательно для заполнения (см. набор параметров его конструктора);
- `DateTimeField` — поле для хранения отметки даты и времени. Присвоив параметру `auto_now_add` конструктора значение `True`, мы предпишем Django при создании новой записи записывать в это поле текущие дату и время. А параметр `db_index` при присваивании ему значения `True` укажет создать для этого поля индекс (при выводе объявлений мы будем сортировать их по убыванию даты публикации, и индекс здесь очень пригодится).

Вот в чем основная прелесть Django и его механизма моделей: во-первых, мы описываем структуру таблицы базы данных в понятных нам терминах любимого Python, а во-вторых, описываем на очень высоком уровне, в результате чего нам не придется беспокоиться, скажем, о занесении нужного значения в поле `published` вручную. И это не может не радовать.

Еще один любопытный момент. Практически всегда таблицы баз данных имеют поле для хранения *ключей* — уникальных значений, которые будут однозначно идентифицировать соответствующие записи (*ключевое поле*). Как правило, это поле имеет целочисленный тип и помечено как автоинкрементное — тогда уникальные числовые значения в него будет заносить само программное ядро СУБД. В моделях Django такое поле явно объявлять не надо — фреймворк создаст его самостоятельно.

Сохраним исправленный файл. Сейчас мы сгенерируем на его основе миграцию, которая создаст в базе данных все необходимые структуры.

НА ЗАМЕТКУ

По умолчанию вновь созданный проект Django настроен на использование базы данных в формате SQLite, хранящейся в файле `db.sqlite3` в папке проекта. Эта база данных будет создана уже при первом запуске отладочного веб-сервера.

1.8. Миграции

Миграция — это модуль Python, созданный самим Django на основе определенной модели и предназначенный для формирования в базе данных всех требуемых этой моделью структур: таблиц, полей, индексов, правил и связей. Еще один замечательный инструмент фреймворка, заметно упрощающий жизнь программистам.

Для формирования миграции на основе модели `bb` мы переключимся в командную строку, проверим, остановлен ли отладочный веб-сервер и находимся ли мы в папке проекта, и дадим команду:

```
manage.py makemigrations bboard
```

Команда `makemigrations` утилиты `manage.py` запускает генерирование миграций для всех моделей, объявленных в приложении, чье имя записано после самой команды, и не изменившихся с момента предыдущего генерирования миграций.

Сформированные таким образом модули с миграциями сохраняются в пакете `migrations`, находящемся в пакете приложения. Модуль с кодом нашей первой

миграции будет иметь имя 0001_initial.py. Откроем его в текстовом редакторе и посмотрим на хранящийся в нем код (листинг 1.7).

Листинг 1.7. Код миграции, создающей структуры для модели Bb (приводится с незначительными сокращениями)

```
from django.db import migrations, models

class Migration(migrations.Migration):
    initial = True
    dependencies = [ ]

    operations = [
        migrations.CreateModel(
            name='Bb',
            fields=[
                ('id', models.AutoField(auto_created=True,
                    primary_key=True, serialize=False, verbose_name='ID')),
                ('title', models.CharField(max_length=50)),
                ('content', models.TextField(blank=True, null=True)),
                ('price', models.FloatField(blank=True, null=True)),
                ('published', models.DateTimeField(auto_now_add=True,
                    db_index=True)),
            ],
        ),
    ]
```

Описание средств Django, применяемых для программирования миграций вручную, выходит за рамки этой книги. Однако приведенный здесь код вполне понятен и напоминает код написанной нами ранее модели. В частности, сразу можно видеть список полей, которые должны быть созданы в таблице базы данных для того, чтобы модель Bb смогла использовать ее для хранения сущностей. Также можно догадаться, что таблица базы данных будет иметь имя Bb — как и модель.

И наконец, первым же элементом списка создаваемых полей идет автоинкрементное ключевое поле id. Мы не объявили это поле в модели явно, и Django создал его самостоятельно для своих собственных нужд.

Миграция при выполнении порождает команды на языке SQL, которые будут отправлены СУБД и, собственно, выполнят все действия по созданию необходимых структур данных. Давайте посмотрим на результирующий SQL-код нашей миграции, отдав команду:

```
manage.py sqlmigrate bboard 0001
```

После команды sqlmigrate, выводящей на экран SQL-код, мы поставили имя приложения и числовую часть имени модуля с миграцией. Прямо в командной строке мы получим такой результат (код для удобства чтения был переформатирован):

```
BEGIN;
--
-- Create model Bb
--
CREATE TABLE "bboard_bb" (
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
    "title" varchar(50) NOT NULL,
    "content" text NULL,
    "price" real NULL,
    "published" datetime NOT NULL
);
CREATE INDEX "bboard_bb_published_58fde1b5" ON "bboard_bb" ("published");
COMMIT;
```

Этот код был сгенерирован для СУБД SQLite (вспомним — проект Django по умолчанию использует базу данных этого формата). Если используется другая СУБД, результирующий SQL-код будет соответственно отличаться.

Что ж, налюбовавшись на нашу первую миграцию, давайте выполним ее. В процессе *выполнения* миграция создает все описанные в ней структуры.

Первое выполнение миграций рекомендуется производить для всех приложений, входящих в проект. Дело в том, что многие приложения, входящие в состав Django, содержат свои миграции, создающие в базе данных структуры для моделей, которые используются в этих приложениях. Так что, если эти миграции не выполнить, приложения окажутся неработоспособными.

Наберем в командной строке команду выполнения всех миграций всех приложений проекта:

```
manage.py migrate
```

Судя по выводимым в командной строке сообщениям, таковых миграций много — десятка два, и они заметно увеличивают объем базы данных. Поэтому при программировании реальных сайтов настоятельно рекомендуется исключать ненужные стандартные приложения из списка зарегистрированных в проекте сразу же (эти приложения будут рассмотрены в *главе 3*).

1.9. Консоль Django

Итак, у нас есть готовая модель для хранения объявлений. Но пока что нет ни одного объявления. Давайте создадим парочку для целей отладки.

Фреймворк включает в свой состав собственную редакцию консоли Python Shell, называемую *консолью Django*. От аналогичной командной среды Python она отличается тем, что в ней в состав путей поиска модулей добавляется путь к папке проекта, в которой запущена эта консоль.

В командной строке наберем команду для запуска консоли Django:

```
manage.py shell
```

И сразу увидим знакомое приглашение `>>>`, предлагающее нам ввести какое-либо выражение Python и получить результат его выполнения.

1.10. Работа с моделями

Не медля, создадим первое объявление — первую запись модели `Bb`:

```
>>> from bboard.models import Bb
>>> b1 = Bb(title='Дача', content='Общество "Двухэтажники". ' + \
'Dва этажа, кирпич, свет, газ, канализация', price=500000)
```

Запись модели создается аналогично экземпляру любого другого класса — вызовом конструктора. Значения полей создаваемой записи можно указать в вызове конструктора посредством именованных параметров, чьи имена совпадают с именами соответствующих полей.

Созданная таким образом запись модели не сохраняется в базе данных, а существует только в оперативной памяти. Чтобы сохранить ее, достаточно вызвать у нее метод `save()` без параметров:

```
>>> b1.save()
```

Проверим, сохранилось ли наше первое объявление, получив значение ключевого поля:

```
>>> b1.pk
1
```

Отлично! Сохранилось.

Атрибут класса `pk`, поддерживаемый всеми моделями, хранит значение ключевого поля текущей записи. А это значение может быть получено только после того, как запись модели успешно сохранится в базе данных.

Мы можем обратиться к любому полю записи, воспользовавшись соответствующим ему атрибутом класса модели:

```
>>> b1.title
'Дача'
>>> b1.content
'Общество "Двухэтажники". Два этажа, кирпич, свет, газ, канализация'
>>> b1.price
500000
>>> b1.published
datetime.datetime(2018, 5, 30, 13, 19, 41, 904710, tzinfo=<UTC>)
>>> b1.id
1
```

В последнем случае мы обратились непосредственно к ключевому полю `id`.

Создадим еще одно объявление:

```
>>> b2 = Bb()
>>> b2.title = 'Автомобиль'
```

```
>>> b2.content = '"Жигули"'
>>> b2.save()
>>> b2.pk
2
```

Да, можно поступить и так: создать «пустую» запись модели, записав вызов конструктора ее класса без параметров и занеся нужные значения в поля позже.

Что-то во втором объявлении маловато информации о продаваемой машине... Давайте дополним ее:

```
>>> b2.content = '"Жигули", 1980 года, ржавая, некрашеная, сильно битая'
>>> b2.save()
>>> b2.content
'"Жигули", 1980 года, ржавая, некрашеная, сильно битая'
```

И добавим еще одно объявление:

```
>>> Bb.objects.create(title='Дом', content='Трехэтажный, кирпич',
price=50000000)
<Bb: Bb object (3)>
```

Все классы моделей поддерживают атрибут класса `objects`. Он хранит *диспетчер записей* — особую структуру, позволяющую манипулировать всей совокупностью имеющихся в модели записей. Диспетчер записей представляется экземпляром класса `Manager`.

Метод `create()` диспетчера записей создает новую запись модели, принимая в качестве набора именованных параметров значения ее полей. При этом он сразу же сохраняет созданную запись и возвращает ее в качестве результата.

Давайте выведем ключи и заголовки всех имеющихся в модели `Bb` объявлений:

```
>>> for b in Bb.objects.all():
...     print(b.pk, ': ', b.title)
...
1 : Дача
2 : Автомобиль
3 : Дом
```

Метод `all()` диспетчера записей возвращает так называемый *набор записей* — последовательность, содержащую записи модели, в нашем случае — все, что есть в базе данных. Сам набор записей представляется экземпляром класса `QuerySet`, а отдельные записи — экземплярами соответствующего класса модели. Поскольку набор записей является последовательностью и поддерживает итерационный протокол, мы можем перебрать его в цикле.

Отсортируем записи модели по заголовку:

```
>>> for b in Bb.objects.order_by('title'):
...     print(b.pk, ': ', b.title)
...
```

```
2 : Автомобиль
1 : Дача
3 : Дом
```

Метод `order_by()` диспетчера записей сортирует записи по значению поля, чье имя указано в параметре, и сразу же возвращает получившийся в результате сортировки набор записей.

Извлечем объявления о продаже домов:

```
>>> for b in Bb.objects.filter(title='Дом'):
...     print(b.pk, ': ', b.title)
...
3 : Дом
```

Метод `filter()` диспетчера записей выполняет фильтрацию записей по заданным критериям. В частности, чтобы получить только записи, у которых определенное поле содержит заданное значение, следует указать в вызове этого метода именованный параметр, чье имя совпадает с именем поля, и присвоить ему значение, которое должно содержаться в указанном поле. Метод возвращает другой диспетчер записей, содержащий только отфильтрованные записи.

Объявление о продаже автомобиля имеет ключ 2. Отыщем его:

```
>>> b = Bb.objects.get(pk=2)
>>> b.title
'Автомобиль '
>>> b.content
'"Жигули", 1980 года, ржавая, некрашенная, сильно битая'
```

Метод `get()` диспетчера записей имеет то же назначение, что и метод `filter()`, и вызывается аналогичным образом. Однако он ищет не все записи, подходящие под заданные критерии, а лишь одну и возвращает ее в качестве результата. К тому же, он работает быстрее метода `filter()`.

Давайте удалим это ржавое позорище:

```
>>> b.delete()
(1, {'bboard.Bb': 1})
```

Метод `delete()` модели, как уже понятно, удаляет текущую запись и возвращает сведения о количестве удаленных записей, обычно малополезные.

Ладно, хватит пока! Выйдем из консоли Django привычным нам способом — набрав команду `exit()`.

И займемся контроллером `index()`. Сделаем так, чтобы он выводил список объявлений, отсортированный по убыванию даты их публикации.

Откроем модуль `views.py` пакета приложения `bboard` и исправим хранящийся в нем код согласно листингу 1.8.

Листинг 1.8. Код модуля views.py пакета приложения bboard

```
from django.http import HttpResponse

from .models import Bb

def index(request):
    s = 'Список объявлений\r\n\r\n\r\n'
    for bb in Bb.objects.order_by('-published'):
        s += bb.title + '\r\n' + bb.content + '\r\n\r\n'
    return HttpResponse(s, content_type='text/plain; charset=utf-8')
```

Чтобы отсортировать объявления по убыванию даты их публикации, мы в вызове метода `order_by()` диспетчера записей предварили имя поля `published` символом «минус». Список объявлений мы представили в виде обычного текста, разбитого на строки последовательностью специальных символов возврата каретки и перевода строки: `\r\n`.

При создании экземпляра класса `HttpResponse`, представляющего отсылаемый клиенту ответ, мы в именованном параметре `content_type` конструктора указали тип отправляемых данных: обычный текст, набранный в кодировке UTF-8 (если мы этого не сделаем, веб-обозреватель посчитает текст HTML-кодом и выведет его одной строкой, скорее всего, в нечитаемом виде).

Сохраним исправленный файл и запустим отладочный веб-сервер. На рис. 1.3 показан результат наших столь долгих трудов. Теперь наш сайт стал больше похож на доску объявлений.

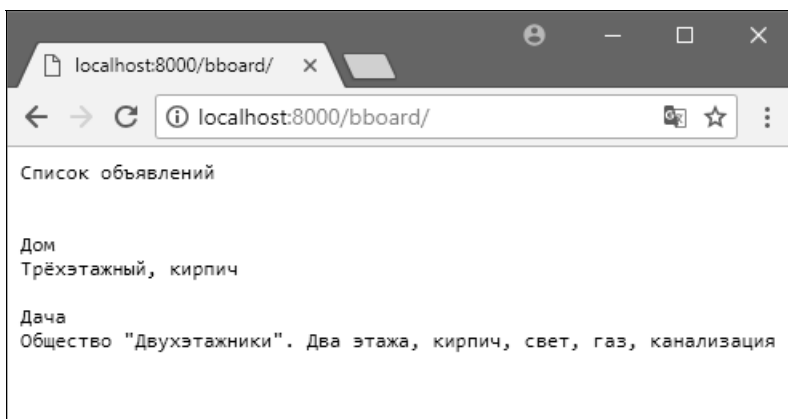


Рис. 1.3. Вывод списка объявлений в виде обычного текста

Аналогичным способом мы можем сформировать не текстовый документ, а полноценную веб-страницу. Вот только писать программу, формирующую строку с HTML-кодом, и, в особенности, отлаживать ее чрезвычайно трудоемко.

Есть ли другой способ? Безусловно, есть. Мы можем использовать шаблоны.