электронный научно-технический журнал
ИНЖЕНЕРНЫЙ ВЕСТНИК
Издатель: Общероссийская общественная организация "Академия инженерных наук им. А.М. Прохорова". Эл No. ФС77-51036. ISSN 2307-0595

Современные средства веб-разработки с использованием фреймворков на примере Django и Laravel

 # 02, июнь 2018

Каяшова Е., Марков И.

УДК: 004.432

Россия, МГТУ им. Н.Э. Баумана
kayashovak@gmail.com
anmarkov1997@gmail.com

Введение

Фреймворк – это программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта. Эта платформа подходит для создания сайтов, бизнес-приложений и веб-сер-висов.

По сути, фреймворк — это множество конкретных и абстрактных классов, а также определений способов их взаимоотношения. Конкретные классы обычно реализуют взаимные отношения между собой, а абстрактные классы представляют собой точки расширения, в которых каркасы могут быть использованы или адаптированы. Для обеспечения расширения возможностей обычно используется объектно-ориентированное программирование (например, части приложения могут наследоваться от базовых классов).

Большинство фреймворков предлагают свою системную архитектуру при создании нового проекта, обычно поддерживающую стандарты MVC (Model-View-Controller), в которой все объекты, относящиеся к одному из трех классов объединены в группу, и каждая из этих групп занимает свое место в файловой системе. Некоторые фреймворки задают такую структуру более жестко, другие не навязывают ее, позволяя разработчику перемещать компоненты приложения по своему усмотрению. Однако большинство разработчиков не меняют основной каркас системной архитектуры, добавляя лишь необходимые в конкретном приложении компоненты. Это позволяет избавиться от создания рутинных элементов приложения и сконцентрироваться исключительно на разработке его логики.

Для поддержания такой структуры и организации ее работы многие фреймворки используют единую точку входа, после чего управление передается дальше.

Также большинство фреймворков позволяют разработчику не задумываться о необходимости настройки сервера и базы данных (БД), или, если это не так, то настройка значительно упрощается и сводится к дополнению или изменению одного, или двух файлов, встроенных в структуру проекта.

Фреймворк может включать вспомогательные программы, библиотеки кода, язык сценариев и другое ПО, облегчающее и ускоряющее разработку и объединение разных компонентов большого программного проекта, а также обеспечивающее защиту приложения от внешних атак. Кроме того, он значительно облегчает интеграцию внешнего кода в проект, если это необходимо.

На сегодняшний день наиболее популярными языками программирования при создании веб-приложений являются PHP и Python.

Как правило, большинство сайтов и веб-сервисов работают на стандартном протоколе HTTP или HTTPS, взаимодействую с базой данных и реализуют внутреннюю логику. Очевидно, что существует достаточное количество разнообразных инструментов и фреймворков, работающих по такому принципу, однако наиболее востребованными и часто используемыми являются Django и Laravel для Python и PHP соответственно.

Отметим, что оба фреймворка являются бесплатными и open-source (полностью открыты для сторонних разработчиков). На сегодняшний день это говорит о том, что они постоянно развиваются и поддерживаются.

Рассмотрим положительные и отрицательные стороны каждого и проведем сравнительный анализ между ними, т.к. их используют при разработке довольно схожих проектов.

Объектом для проведения анализа выбрано несложное веб-приложение, реализующее работу личного блога: авторизация, создание записи, добавление к ней описания, картинки и добавление комментариев. Будем придерживаться традиционного архитектурного стиля REST.

Инфологическая модель для рассматриваемого далее примера представлена на рис. 1. Как видно из UML-диаграммы, между сущностями users (пользователь) и posts (записи) реализована связь «один-ко-многим», также, как и между posts и comments (комментарии).

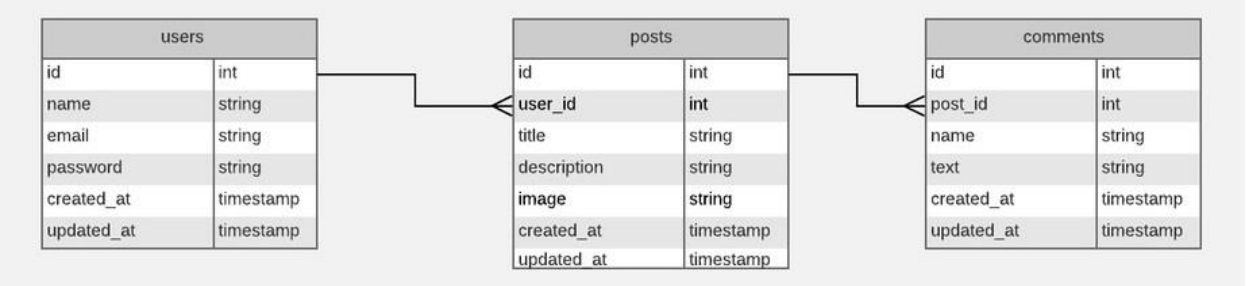

Рис. 1. Инфологическая модель

Первой задачей и поводом для сравнения встает разработка архитектуры приложения. Основная цель здесь показать какие принципы построения приложения предлагает каждый фреймворк.

DJANGO. Django пропагандирует свободное связывание и строгое разделение основных частей приложения. Главным преимуществом такого подхода является возможность внесения изменений в одну часть проекта без изменения другой. Логика работы Django поддерживает стандарты MVC, в которых:

- M – логика доступа к данным, обрабатывает слоем работы с базой данных;
- V – определяет какие данные получать и как их отображать, образуется представлениями и шаблонами;
- C – выбирает представление в зависимости от действий пользователя, вызывает функцию обработки по входящему URL.

Однако здесь «C» реализуется больше средой разработки, поэтому наиболее интересные действия происходят в моделях, представлениях и шаблонах. Из-за этого факта многие относят Django к модели MTV:

- M (Model) – доступ к данным. Знает, как получить, как проверить, как связать данные между собой;
- T (Template) – принимает решение относительно представления данных – как и что отображается;
- V (View) – слой представления, бизнес-логики. Принимает решение, какие данные получать и какие шаблоны к ним применять. Своего рода мост между моделью и шаблоном.

Основываясь на принципе разделения приложения на независимые малые части, архитектура нашего приложения выглядит таким образом (рис.2).

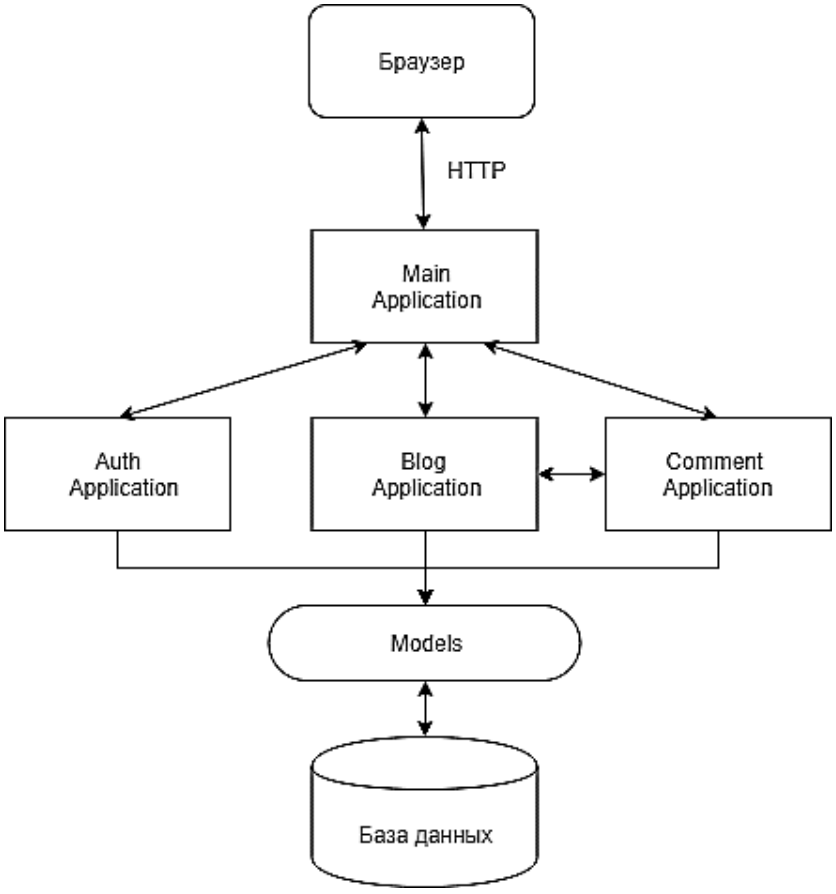

Рис. 2. Архитектура приложения в Django.

Main Application реализует маршрутизацию к приложениям Auth, Blog и Comment в зависимости от действий пользователя, которые потом будут работать базой данных.

В свою очередь эти приложения выполняют свою индивидуальную логику: авторизацию, создание и редактирование записи и добавление комментариев.

При начале работы средствами фреймворка разработчику автоматические создается панель администрирования, которая позволяет через графический интерфейс вносить изменения в каждое независимое приложение, а именно управлять моделями данных, которые указаны в каждом приложении.

В свою очередь каждое независимое приложение имеет изначально одинаковую структуру, исключением является MainApplication.

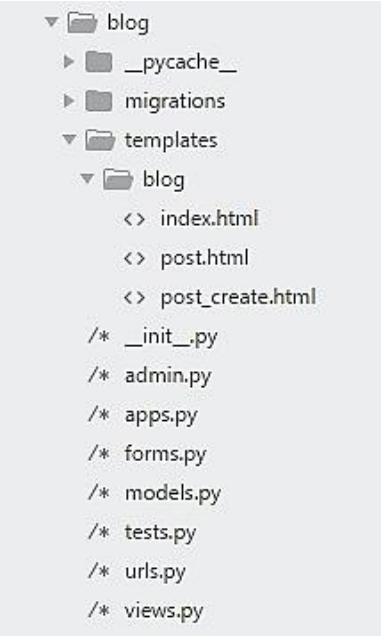

Рис. 3. Структура DjangoApplication

На примере BlogApplication опишем структуру типового приложения:

migrations– хранит журнал миграций в базе данных;

templates – шаблоны представлений в виде файлов разметки;

admin – управление отображением в панели администратора;

apps – управление конфигурацией приложения;

forms – работа с формами, применительно к данным;

models – описывает наши модели для работы с базой данных;

urls – назначение обработчиков для входящих URL;

views – описывает обработчики из urls и связывает models с templates.

В Main Application присутствует информация о глобальных настройках и зависимостях всего приложения, файл маршрутизации, подобный urls и файл, необходимый для реализации интерфейса WSGI.

Жизненный цикл запроса представлен следующим образом. Все входящие запросы первоначально попадают в Main Application и в соответствии с маршрутизатором urls.py передаются для обработки контроллером, либо управление переходит Application более низкой иерархии. После попадания в контроллер в нем вступает в работу механизм отображения результата, использующий шаблоны и модели для работы с данными. Само отображение

может наследоваться от различных view, используя их свойства, также в механизме представлений реализован механизм защиты от межсайтовых запросов CSRF при помощи csrf-токена, принципы работы которого рассмотрим чуть позже.

LARAVEL. Laravel – бесплатный PHP-фреймворк, предназначенный для разработки приложений любой сложности. Так же, как и многие современные фреймворки, Laravel использует архитектуру модели MVC, описанную выше.

Кроме того, в Laravel используются Routes для сопоставления URL-адресов и вызываемых контроллеров.

Фреймворк Laravel работает совместно с Composer - менеджер зависимостей для PHP. Он содержит множество PHP-пакетов от сообщества разработчиков Laravel, поддерживает версионность, а также обрабатывает автозагрузку PHP-классов.

Структура любого нового проекта Laravel 5.3 представлена на рис. 4.

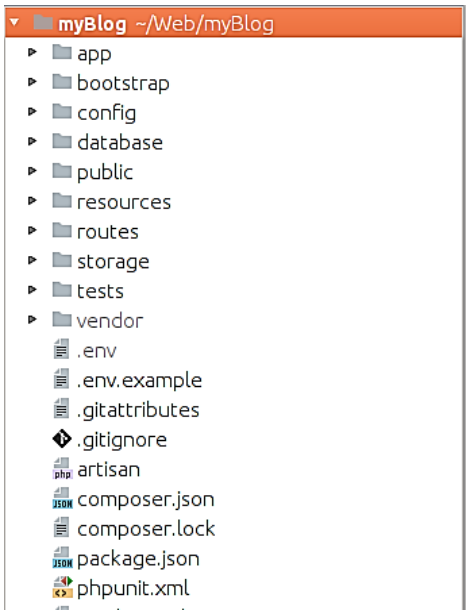

Рис. 4. Структура проекта на Laravel

Фреймворк, однако, является гибким и не накладывает почти никаких ограничений на то, где находится любой данный класс до тех пор, пока Composer может загрузить его автоматически. Каждая папка несет отдельную смысловую нагрузку, т.е.:

app – содержит основной код приложения: контроллеры, модели, сервисы, middleware, провайдеры и т.д.;

bootstrap – содержит несколько файлов, загружающих платформу и настраивающих автоматическую загрузку;

config – содержит файлы конфигурации;

database – содержит миграции базы данных и сидеры (классы для наполнения БД начальными данными);

public – содержит файл index.php, который является «входной точкой» для всех запросов, поступающих в приложение. Также эта папка содержит ресурсы, такие как изображения, JavaScript, CSS;

resources – содержит представления, а также нескомпилированные ресурсы;

routes – содержит файлы маршрутизации;

storage – содержит файлы генерируемые приложением, кэш и файлы журнала приложения;

tests – содержит файлы тестов;

.env – файл который хранит всю секретную информацию проекта: данные для доступа к БД и другие вещи. Организован в виде ключ-значение.

Это позволяет сделать структуру приложения более удобочитаемой и понятной.

Учитывая особенности фреймворка Laravel, архитектура нашего приложения будет иметь вид, показанный на рисунке 5.

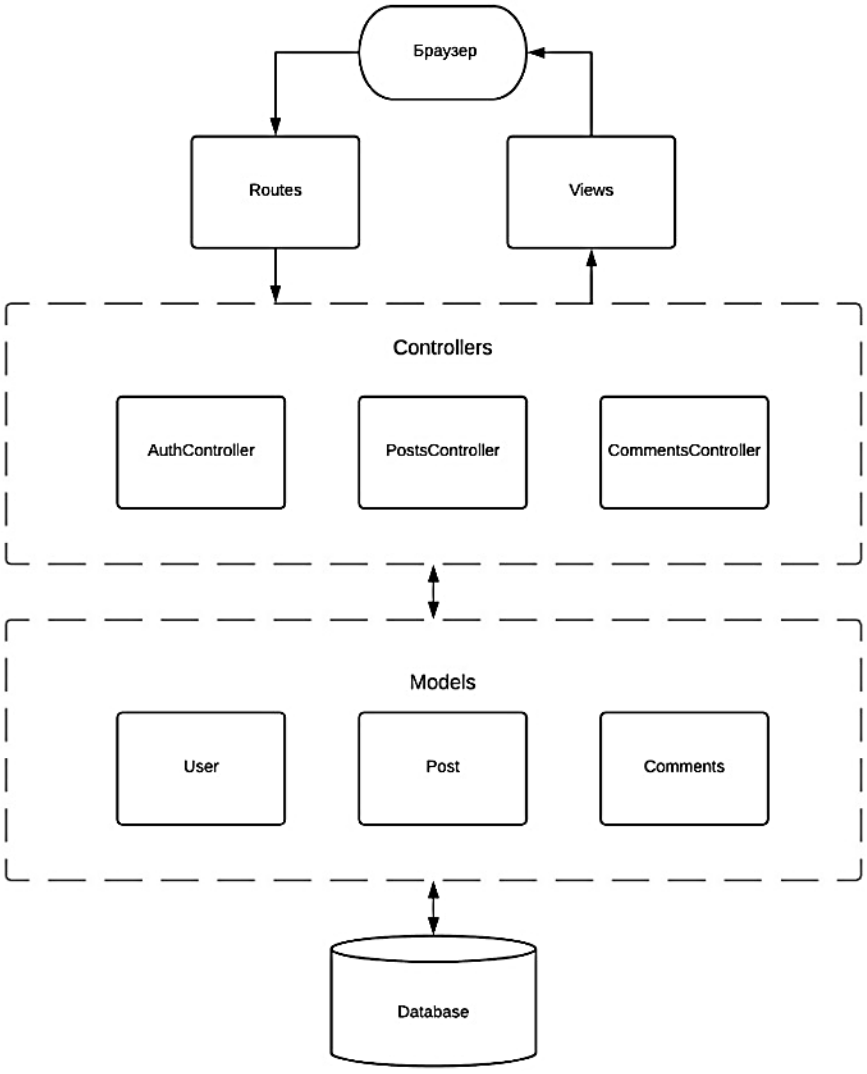

Рис. 5. Архитектура приложения в Laravel

Рассмотрим подробнее жизненный цикл запроса. Как и в большинстве фреймворков, Laravel использует единую «точку входа» – файл public/index.php. Все запросы к серверу перенаправляются в этот файл настройками веб-сервера (Apache/Nginx), который далее подключает необходимые ресурсы для работы приложения и перенаправляет вызов либо в

HTTP-ядро, либо в ядро консоли, в зависимости от типа запроса. Эти два ядра служат центральным местом, через которое протекают все запросы. HTTP-ядро определяет массив загрузчиков, которые настраивают обработку ошибок, ведение журналов, определяют среду приложения и список HTTP посредников, через которые должны пройти все запросы, прежде чем будут обработаны приложением. К таким посредникам можно отнести сервис-провайдеры, отвечающие за начальную загрузку всевозможных компонентов фреймворка: таких как БД, очередь, проверка ввода и маршрутизация. Одна из основных функций HTTP-посредников – CSRF защита или защита от атак с подделкой межсайтовых запросов, она обеспечивается посредством проверки CSRF-токена, о логику работы которого рассмотрим позже. После регистрации всех сервис-провайдеров и загрузки приложения запрос поступает в роутер для обработки.

Порог вхождения. Под порогом вхождения будем понимать минимальный набор действий для того, чтобы приложение начало работать с пользователем. В обоих фреймворках порог вхождения находится приблизительно на одном уровне. Так для отображения собственной HTML-страницы необходимо описать маршрутизатор до указания конечного контроллера обработки запроса, описать контроллер, в котором будет происходить логика отображения и настроить файл конфигурации.

Оценка архитектуры проекта. Оба фреймворка поддерживают стандарты MVC, однако Django пропагандирует разделение проекта на отдельные приложения, таким образом проект состоит из множества таких приложений, и разработка каждого из них может вестись независимо. Laravel создает единую системную структуру, в которой компоненты MVC переплетены между собой теснее, поэтому задача модульности проекта возлагается на плечи разработчика.

1. Основы работы фреймворков

1.1. Механизм маршрутизации

Задача маршрутизатора указывать доступные для обработки входящие запросы от пользователя и назначать им обработчики. Оба механизма достаточно похожи, однако Laravel позволяет более детально на этапе маршрутизации работать в стиле архитектуры REST с дополнительным указанием http-методов помимо шаблона строки запроса. Django может отправлять запрос из основного приложения в маршрутизаторы более низкой иерархии, реализуя модульность проекта. Оба инструмента также позволяют задать имя каждому маршруту, чтобы его вызов внутри проекта осуществлялся проще.

В обоих фреймворках маршрутизация настраивается легко и удобно. При этом, с изменением расположения проекта или переносом его на новый сервер разработчику не нужно каждый раз указывать полный путь до управляющего скрипта. Фреймворк позаботится об этом самостоятельно, достаточно указать, где в предложенной системной архитектуре проекта он находится.

1.2. Работа с данными

Достаточно популярным решением при работе с данными стало использование внутренней ORM (Object Related Model), работа которой заключается в повышении удобства работы с базой данных через объекты-классы, описывающие существующие сущности в базе данных. Также такой подход снижает риск возникновения SQL-инъекций и позволяет загружать данные из нескольких таблиц (решая проблему N+1) или же обрабатывать данные из БД частями. Такой механизм работы предусмотрен в обоих фреймворках, но прежде чем описывать классы-объекты, в Django и Laravel сначала необходимо настроить работу с БД и описать миграции.

Как было указано ранее, настройка информации о БД в Laravel описывается в файле .env, достаточно лишь изменить пару строчек. В Django аналогичная настройка происходит в глобальном файле настроек settings.

Миграции - это набор классов, позволяющих описать базу данных, создавать или удалять таблицы из базы данных или вносить изменения в них. Они подобны контролю версий для базы данных, что позволяет легко работать с ней совместно. Для внесения изменений в базу данных внутри классов миграций создаются методы, описывающие эти изменения автоматически.

Для описания моделей необходимо не только указать, с какой таблицей она связана, но и прописать внутренние связи между моделями, аналогичные связям между таблицами. И в Django и в Laravel это делается достаточно просто. В Laravel, каждая новая модель наследует класс Model, в которой описаны все возможные связи в виде методов. В модели нужно создать функцию, внутри которого вызывается метод класса Model, соответствующий определенной связи и в качестве параметра указать модель, с которой эта связь осуществляется. В Django каждая модель является расширением класса models.Model, при её описании указываются поля и свойства с использованием классов из models.* и написанием собственных функций при необходимости.

Кроме того, поддерживая парадигму объектно-ориентированного программирования, в модели могут быть добавлены методы и переменные, определяющие ее. Так, например, получая экземпляр из базы данных с помощью модели, можно получать значения её полей аналогично атрибутам класса.

В целом настройка работы с базой данных, производится достаточно быстро, а изменения в ее структуру вносятся легко, благодаря гибкости алгоритмов работы ORM.

1.3. Представление

Представление можно понимать, как заполненный данными HTML-шаблон, который видит пользователь в результате отправления запроса. Какой шаблон выбирается, и, какими данными заполняется, решает контроллер. Вместе с вызовом шаблона, контроллер передает в него необходимые переменные. В Laravel и Dajngo в качестве переменных могут высту-

пать данные моделей, что позволяет динамически получать информацию из БД о соответствующем объекте внутри шаблона. Каждый шаблон имеет внутреннюю логику отображения с использованием операторов и конструкций, схожих с языками программирования.

В Laravel также есть возможность создать единый каркас для шаблонов, который представляет дизайн проекта. Так, например, если header и footer одинаковые на каждой странице, то необязательно копировать их в каждый шаблон. Можно создать их в виде двух отдельных шаблонов и вызывать внутри новых, в том месте, где это необходимо. В Django на такие случаи предусмотрен механизм наследования шаблонов. Это позволяет описать базовый шаблон в виде контейнера, который содержит в себе общую для всех дочерних шаблонов: разметку страницы, свойства. Далее каждый новый шаблон, наследуюсь от базового, содержит его свойства и разметку, которые может дополнить собственными.

В обоих фреймворках код веб-проекта, CSS, JS, HTML-код страниц разделены на отдельные директории. Laravel использует шаблонизатор Blade, который позволяет отделить вёрстку от PHP-кода. Django использует свой язык шаблонов для внедрения переменных и операторов в разметку.

Результаты заполнения шаблонов представлены на рис. 6 и рис. 7.

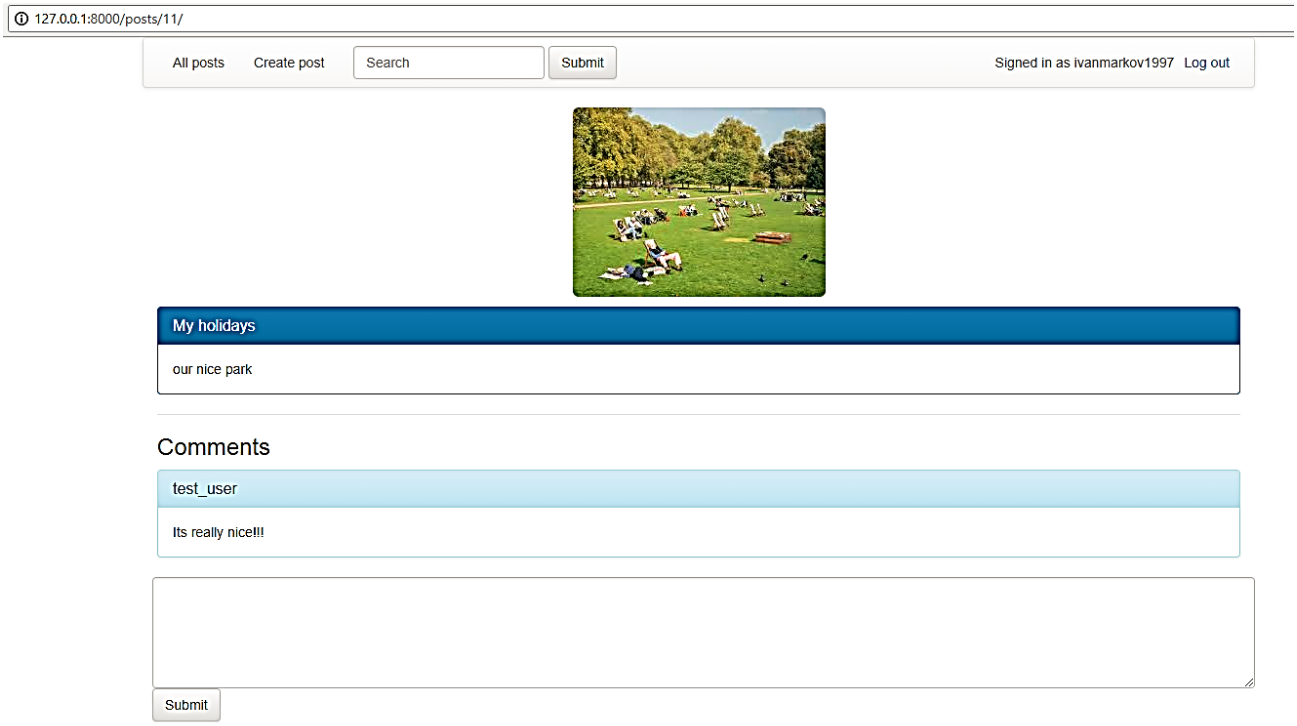

Рис. 6. Просмотр поста и добавление комментария (Django)

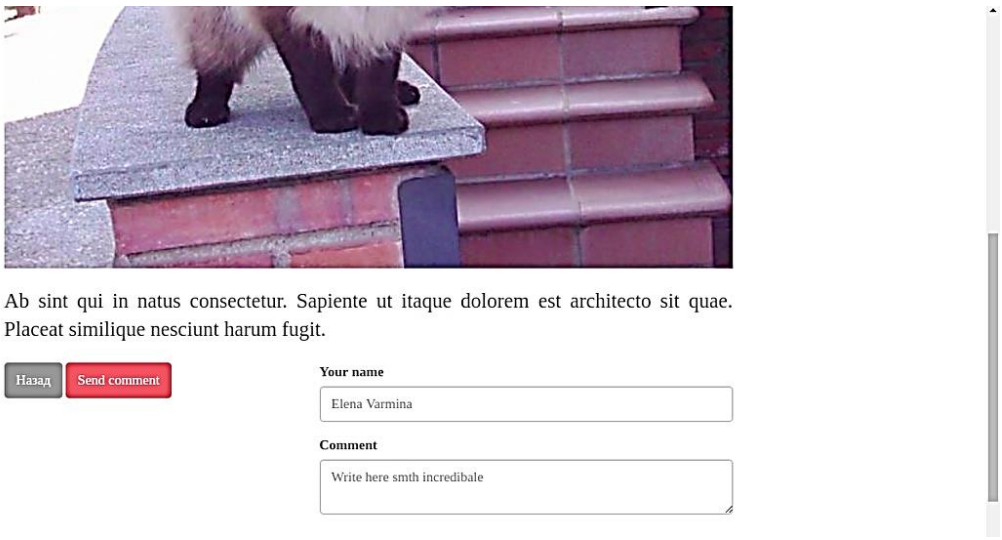

Рис. 7. Просмотр поста и добавление комментария (Laravel)

1.4. Дополнительные возможности

В Django предусмотрена встроенная панель администрирования. Это является крайне удобным дополнением, позволяющим быстро манипулировать данными, пользователями и т.д., не прибегая к написанию сложных скриптов и создания собственного интерфейса. Типовыми функциями панели являются создание, удаление и редактирование данных, привязанных с моделями и пользователей системы. В Laravel панель администратора создается разработчиком самостоятельно. Но для тестирования работы приложения, Laravel предоставляет Seeder, который заполняет базу нужным количеством тестовых данных.

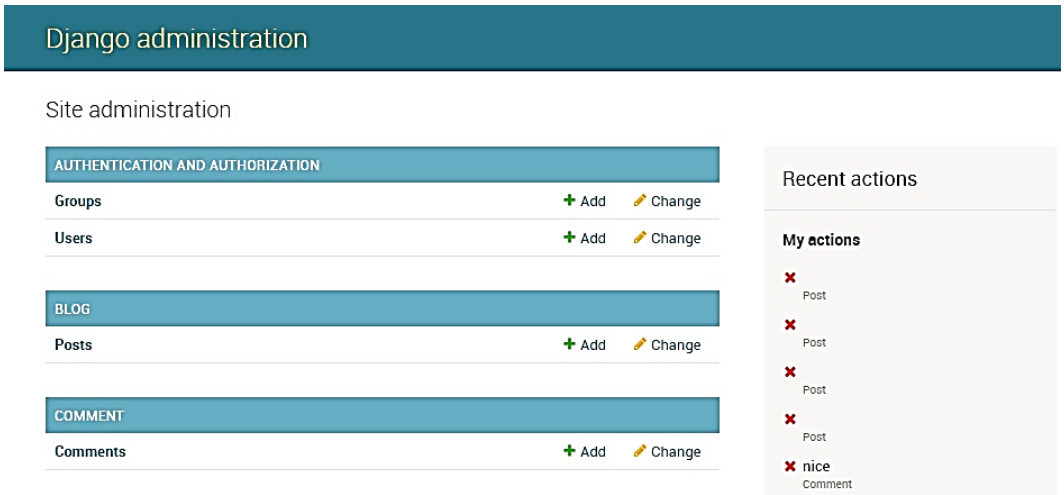

Рис. 8. Панель администрирования Django

Оба фреймворка предоставляют разработчику возможность не задумываясь работать с сессиями. В языке Python сессий нет в принципе, но эту проблему решает программное обеспечение фреймворка Django. Для работы с сессиями используют Middleware — промежуточное программное обеспеченье, имеющее доступ к объекту запроса и объекту ответа.

Таким образом, необходимо просто нужным образом настроить Middleware. Такой механизм также значительно облегчает процесс авторизации, где фреймворк сам формирует

токен, по которому осуществляется проверка пользователя, и проверяет его верность и актуальность при каждом обращении пользователя к приложению. Разработчик же просто запускает этот алгоритм.

Laravel позволяет создать простую авторизацию, с помощью логина и пароля. В один клик создаются миграции и модели users, remember_tokens, контроллеры и простые шаблоны, что позволяет значительно сэкономить время на разработке. Стоит правда добавить, что данным механизмом пользуются только начинающие разработчики, так как все чаще применяется двухступенчатая авторизация или проверка логина, за счет отправки email-уведомлений, что не предусмотрено в авторизации «из коробки» в Laravel.

Django и Laravel упрощают защиту приложения от атак вида «межсайтовая подделка запроса» (CSRF - Cross Site Request Forgery). Этот тип атак случается, когда злонамеренный Web сайт содержит ссылку, кнопку формы или некоторый javascript, который предназначен для выполнения некоторых действий на вашем Web сайте, используя учетные данные авторизованного пользователя, который посещал злонамеренный сайт в своем браузере. CSRF - это гарантирование того, что GET запросы (и другие “безопасные” методы) свободны от побочных эффектов. Запросы через “небезопасные” методы, такие как POST, PUT и DELETE могут быть защищены с подключением нужного Middleware и добавлением CSRF-токена внутрь формы отправки запроса от пользователя. Токен гарантирует, что ваши данные не будут отправлены на внешние ресурсы. Однако не стоит добавлять токен при обращении через формы на URL сторонних сайтов. Это может привести к утечке токена. В Laravel и Django необходимо указывать скрытое поле CSRF-токена каждой форме шаблона, чтобы Middleware могло проверить запрос.

1.5. Настройки приложения

Как упоминалось ранее, многие функции и настройки приложения Django и Laravel описываются в файле глобальных настроек проекта settings и env соответственно. Файл настройки достаточно важная часть любого фреймворка, потому что именно в нем указываются внешние зависимости, информация о БД и многое другое, что позволяет ориентировать ваше приложение на индивидуальный механизм работы уже на этапе создания проекта.

Опишем основные секции, на которые стоит обратить внимание.

Core settings. Самая обширная секция. К основным её настройкам относятся:

- описание администраторов;
- разрешенные хосты для обращения к приложению;
- механизм кэширования;
- параметры и особенности CSRF-защиты;
- местное время, дата и языки;
- система логирования;
- пути до медиа файлов;
- используемые Middleware;

- SSL-механизмы;
- указание директорий для поиска шаблонов;
- определение базы данных для работы.

Auth settings. Очевидно, что секция описывает авторизацию. Тут можно настроить:

- внутренние механизмы авторизации (типы);
- название модели для работы с пользователем;
- механизмы хэширования паролей.

Session settings. Механизмы работы сессий.

Static settings. Описание путей до хранимых статических файлов (картинки, файлы .js и т.д.).

Варьируя данные параметры можно очень гибко и быстро менять особенности работы приложения, причем они оказывают минимальное влияние на написание кода в иных файлах.

2. Скорость работы

2.1. Дополнительное расширение

Однако, какова цена такой универсальной системы и чем приходится расплачиваться, выбирая фреймворки. Или они являются прямым путем к улучшению работы системы и ее скорости? Почему фреймворки рекомендованы для проектов средней сложности на уровне больших интернет магазинов?

Для ответа на эти вопросы придется рассмотреть использование фреймворка с практической точки зрения. Поскольку фреймворк предоставляет гибкий функционал для проектирования веб-сервиса, добавим дополнительный функционал и сравним удобство работы и скорость работы приложения.

У Django предусмотрен компонент для проектирования приложений в архитектурном стиле REST под названием django-rest. Интегрируем этот компонент в наше приложение для удобного и гибкого преобразования данных модели в формат JSON и обратно. Такая функция достаточно часто используется в проектировании сервисов. Для сравнения скорости работы проделаем перевод в формат JSON классическими средствами Python и с использованием компонента django-rest.

Рассмотрим функционал, реализованный описанными средствами. Для реализации перевода в JSON средствами rest потребовалось минимум усилий. Всё работает достаточно гибко и требует минимум символов для ввода с клавиатуры, в отличие от чистого Python.

Теперь сравним скорость работы для двух случаев.

Таблица 1. Сравнение скорости Django

Django + django-rest	Django + native json
186 ms.	48 ms.

Проведем аналогичный эксперемент с Laravel. Попробуем получить информацию, представленную в формате json, с удаленного сервера средствами чистого PHP и с помощью расширения Guzzle и сравним скорость ответа. Guzzle - это HTTP-клиент PHP, который упрощает отправку HTTP-запросов и тривиальную интеграцию с веб-службами. Он предоставляет простой интерфейс для построения строк запроса, запросов POST, потоковой передачи больших загрузок, может отправлять как синхронные, так и асинхронные запросы с использованием одного и того же интерфейса и т.д.

Как и в Django, Guzzle позволил сократить количество кода, однако результаты получились следующие (табл. 2).

Таблица 2. Сравнение скорости Laravel

Laravel + Guzzle	Laravel + clear PHP
1038 ms.	413 ms.

Вот и ответ. Как позывает эксперимент, время ответа сервера увеличивается с применением стороннего гибкого функционала, в то время как классические средства дают приличное ускорение в работе, однако такой способ более трудоемкий и требует больше времени. Тут можно провести аналогию языками программирования. Чем язык более высокоуровневый, тем он более простой в использовании, но медленнее в работе.

2.2. Скорость ответа Laravel и Django

Мы убедились, что за гибкость и удобство работы с дополнительными расширениями приходится платить производительностью. Поэтому выбор фреймворка при создании приложения достаточно важная проблема и в системах с высокими требованиями к скорости работы стараются применять фреймворки там, где это не повлияет на производительность.

Однако если работать с фреймворком необходимо, то стоит выбрать тот, который меньше всего влияет на скорость работы. Для примера сравним время ответа на авторизацию пользователя через веб-формы, реализованную в нашем приложении на Django и Laravel.

Таблица 3. Сравнение скорости авторизации Laravel и Django

Laravel	Django
355 ms.	231 ms.

Полученные результаты показывают, что работа приложения, реализованного на Django, будет несколько быстрее.

Заключение

Оба фреймворка достаточно похожи друг на друга и пользуются огромной популярностью в мире веб-разработки, не только среди начинающих, но и среди опытных разработчиков. Главными целями их использования может послужить избавление от написания рутинного программного кода для типовых операций и простота манипулирования проектом благодаря встроенным механизмам и инструментам работы. Реализованы в рамках шаблона

проектирования MVC с четким разделением проектам на 3 составляющие части. Для работы с базой данных предусмотрены специализированные модели данных, для определения исполняющего скрипта задействован механизм маршрутизации с описанием возможных шаблонов входящих URL. Отображением данных пользователю реализуется на основе гибких в работе шаблонов HTML-страниц с возможность использования внутренней логики, переменных и защиты от межсайтовых поддельных запросов.

Одним из отличий фреймворков является предусмотренный план архитектуры: у Django рекомендует разделять проект на независимые приложения, Laravel – предоставляет свободу выбора архитектуры. Кроме того, Django предоставляет панель администрирования «из коробки», которая является сильным упрощением при управлении моделям. Уже на этапе создания описания моделей и регистрации их в панели администратора есть возможность работать с данными без описания контроллеров, маршрутизаторов и создания интерфейса администратора. В Laravel панель администратора создается разработчиком самостоятельно исходя из требований к ней разработчика. В Django и Laravel предусмотрена возможность автоматического наполнение БД тестовыми данными. Однако, применительно к Django, такой подход *нельзя* использовать для тестирования приложений, поскольку тестовая среда очищает содержимое тестовой базы данных после каждого теста; как результат, - любые данные добавленные с помощью пользовательских запросов SQL будут утеряны. В свою очередь Django опережает Laravel в производительности в некоторых случаях, что может сыграть важную роль при выборе фреймворка для разработки. В остальных аспектах инструменты имеют схожие принципы работы.

Список литературы

[1]. Джефф Форсье. Django. Разработка веб-приложений на Python: учебник / Джефф Форсье, Пол Биссекс, Уэсли Дж. Чан. Пер. с англ. - М: Символ-Плюс, 2013 – 456 с.

[2]. Django documentation [Электронный ресурс]. Режим доступа: https://docs.djangoproject.com/en/2.0/ (дата обращения: 3.06.2018)

[3]. Django REST Framework [Электронный ресурс]. Режим доступа: http://www.django-rest-framework.org/ (дата обращения: 3.06.2018)

[4]. Laravel документация 5.x [Электронный ресурс]. Режим доступа: https://laravel.ru/docs/v5 (дата обращения: 3.06.2018)

[5]. Laravel. Официальный сайт [Электронный ресурс]. Режим доступа: https://laravel.com/ (дата обращения: 3.06.2018)

[6]. Фреймворки в веб-разработке [Электронный ресурс]. Режим доступа: https://web-creator.ru/articles/about_frameworks (дата обращения: 3.06.2018)