УДК 004.9

MVC-АРХИТКЕТУРА В СОВРЕМЕННЫХ ВЕБ-ФРЕЙМВОРКАХ НА ПРИМЕРЕ LARAVEL

Кудрявцев Д.С., студент группы ПИб-191, IV курс
Ларин Н.М., студент группы ПИб-191, IV курс
Кузбасский государственный технический университет имени Т.Ф. Горбачева
г. Кемерово

В современном мире разработка крупного, многофункционально приложения с трудом представляется без соответствующего фреймворка. Для примера возьмём локальную корпоративную информационную систему для филиала ППК «РОСКАДАСТР» по Кемеровской области – Кузбассу, которую мы разрабатываем в рамках выпускной квалификационной работы. Она является веб-приложением, содержащим авторизацию, права доступа, использует REST-архитектуру [1] и множество других современных технологий. Разработка таковой будет затруднена, если мы решим использовать нативные средства какого-либо языка. Придётся самостоятельно реализовывать сложную структуру из классов, что очень затратно по времени, и под силу не каждому разработчику. Другой вариант – использовать различные библиотеки, что тоже не оптимальное решение, потому что библиотеки могут либо конфликтовать друг с другом, либо вообще использоваться лишь один раз, из-за чего смысл во всём остальном функционале пропадает. Гораздо более логичным решением будет использовать фреймворк – набор готовых решений, предназначенных для самых различных задач.

Множество современных фреймворков для веб-разработки реализуют архитектуру **MVC**, ставшую уже стандартом хорошего браузерного приложения – контроллер получает запрос, осуществляет некоторую логику, обращается к модели, чтобы та, в свою очередь, сделала вытяжку из базы данных. Результат их совместного труда передаётся в представление, которое приводит данные в доступный и понятный пользователю вид и осуществляет их отрисовку (рис. 1).

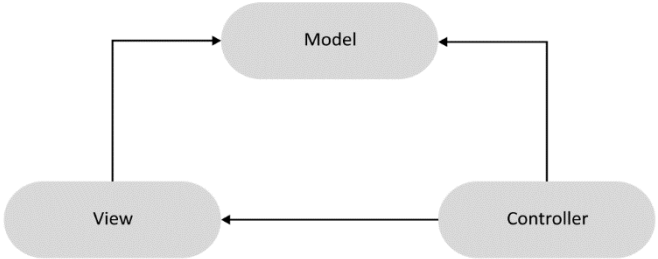

Рисунок 1. Схема взаимодействия составных частей MVC-архитектуры

Это было краткое описание MVC-архитектуры. На самом деле, взаимодействие между составными частями происходит через множество классов-прослоек, которые позволяют добавить дополнительную логику и расширить возможности приложения, при этом сохраняя модульность и поддерживаемость кода и самого проекта (рис. 2).

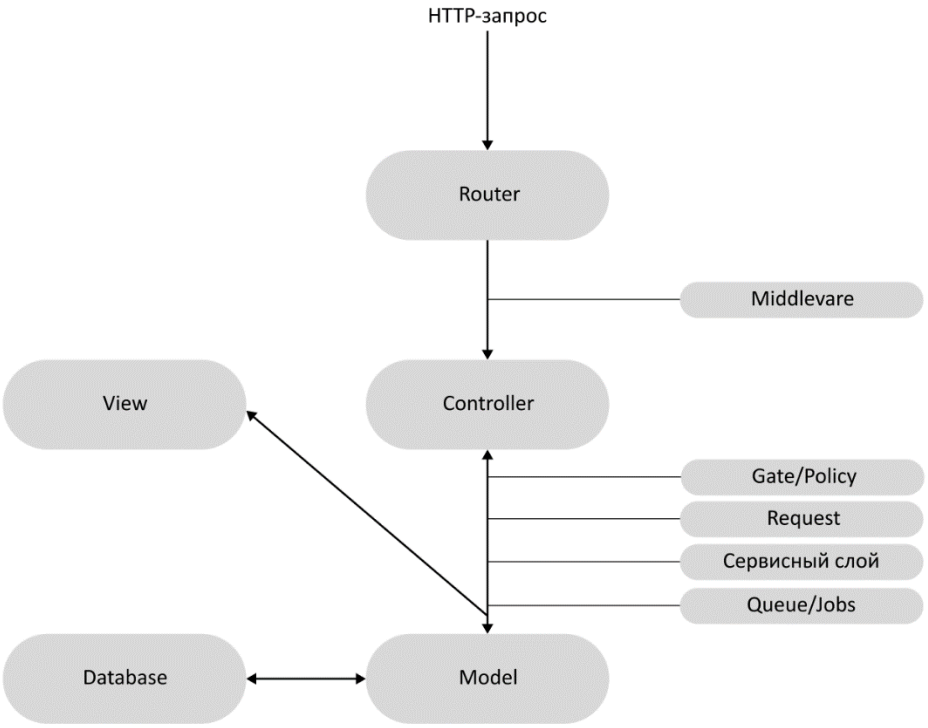

Рисунок 2. Схема взаимодействия составных частей MVC-архитектуры в Laravel

Рассмотрим некоторые классы, доступные во фреймворке Laravel 9 [2]. Однако, прежде чем начать несколько слов о самом Laravel и принципах его работы.

В основе работы Laravel лежат три архитектурные концепции [3] – Service Container, Service Provider и Facade. Service Container предназначен для внедрения зависимостей и реализует инверсию управления. Service Provider позволяет инициализировать некоторые классы на старте приложения, либо самостоятельно фреймворком, позволяя нам работать с ними без лишних строк кода. Например, любые объекты классов будут созданы автоматически, если они были переданы в качестве параметров в метод контроллера. Это значительно упрощает работу с, например, сервисным слоем. И, наконец, Facade (далее фасады) – «статические прокси»-классы, которые могут быть использованы в любой части кода в любой момент. В остальном же Laravel функционирует так же, как и любой другой фреймворк для веб-разработки. Теперь, ознакомившись с принципами работы Laravel, приступим к классам, реализующим MVC-архитектуру, и некоторым прослойкам, которые могут в них использоваться.

Точкой входа в наше приложение является HTTP-запрос, который обрабатывается в файле web.php, либо api.php. В этих файлах с использованием фасада route написаны так называемые «роуты», связывающие между собой запрос и методы контроллера. Именно здесь можно использовать наш первый класс-прослойку – Middleware. Экземпляр класса Middleware содержит в себе некоторую логику, которая осуществляется до того, как будет вызван метод контроллера. Как правило, они используются для проверки прав доступа пользователя к данному методу контроллера.

После того, как роут, соответствующий HTTP-запросу, найден, в дело вступает контроллер. Он взаимодействует с моделью и осуществляет обработку полученных данных, и/или некоторой логики. Как правило, контроллер – точка схода множества классов-прослоек как из фреймворка, так и написанных пользователем. Мы рассмотрим лишь некоторые из множества доступных в Laravel.

Gate/Policy – классы, использующиеся для авторизации, проверки прав доступа и возможностей пользователей.

Request. Базовый объект класса Request содержит данные, пришедшие от клиента, и предоставляет методы для их валидации. Так как проверки одних и тех же данных могут повторяться из метода в метод, Laravel предоставляет возможность создания собственных Request-классов (например, PostRequest), где также можно объявлять правила валидации.

Queue/Jobs – мощный механизм, позволяющий вынести выполнение трудоёмких и долгих операций в фоновый режим.

Сервисный слой (Service) [4] – концепция не Laralel, но PHP, которая заключается в том, что мы сосредотачиваем логику нашего приложения в классах-сервисах, а в своих контроллерах обращаемся к ним. Благодаря автоматический инициализации объектов через параметры методов, мы можем очень легко использовать сервисный слой в Laravel, делая наш код чище и читабельнее, а также действительно соответствующим букве S из SOLID.

Как было описано выше, контроллер для получения некоторых данных обращается к **модели (Model)**. Если обобщить, модель предоставляет набор методов, позволяющих осуществлять вытяжку из базы данных без написания SQL-запросов напрямую. В Laravel для этого существует два мощных механизма: **Query Builder** и **Eloquent ORM**. Они оба могут взаимодействовать с базой данных с одним ключевым отличием, заключающемся в способе взаимодействия с базой данных. Первый отправляет запросы в базу напрямую, а второй использует для этого модель.

Затрагивая тему работы с базами данных средствами MVC-фреймворка, невозможно не упомянуть **миграции (Migrations)**. Миграции по сути являются слепками таблиц базы данных и определяют их схемы. Они что-то наподобие системы контроля версий, позволяющей изменить структуру базы данных, не прибегая к её пересозданию и перезаполнению.

Класс **View** позволяет осуществлять различные взаимодействия с Blade-шаблонами, HTML-файлами, которые могут интегрировать PHP-код

в HTML-разметку, используя короткие директивы. Один из множества вариантов использования View – возможность отправлять некоторые данные сразу в несколько представлений вне контроллера (с использованием Service Provider).

Обобщим полученные знания и рассмотрим механизм MVC-архитектуры внутри Laravel с учётом некоторых упомянутых классов-прослоек:

1. поступление HTTP-запроса;
2. обработка запроса роутером:
   2.1. вызов Middleware перед переходом к методу контроллера;
3. вызов метода контроллера, соответствующего присланному запросу:
   3.1. проверка прав пользователя с помощью Gate;
   3.2. валидация данных с помощью Request;
   3.3. реализация логики, находящейся в сервисном слое;
4. использование методов модели для вытяжки данных из базы данных;
5. передача данных во View;
6. отрисовка HTML-страницы с помощью шаблонизатора Blade.

Таким образом, мы в общих чертах рассмотрели принципы работы MVC-архитектуры во фреймворке Laravel. Наличие большого количества классов-прослоек позволяет не только расширить возможности нашего приложения, но и обеспечить его модульность и расширяемость, а Laravel охотно помогает нам с этим, что не может не радовать. Именно эти возможности фреймворка Laravel использованы нами при разработке корпоративной информационной системы для филиала ППК «РОСКАДАСТР» по Кемеровской области – Кузбассу.

Статья подготовлена под научным руководством профессора кафедры прикладных информационных технологий А.Г. Пимонова.

Список литературы:

1. Коллективный блог habr.com. – URL: https://habr.com/ru/post/38730/ (дата обращения: 24.03.2023).

2. Документация по Laravel 9. – URL: https://laravel.com/docs/9.x (дата обращения: 24.03.2023).

3. Сайт о заметках разработчика dev-notes.ru. – URL: https://www.dev-notes.ru/articles/laravel/dependency-injection-and-service-container/ (дата обращения: 24.03.2023).

4. Коллективный блог habr.com. – URL: https://habr.com/ru/post/547510/ (дата обращения: 24.03.2023).