ОПТИМИЗАЦИЯ МЕТОДОВ РАБОТЫ С ДАННЫМИ ПРИ СОЗДАНИИ ИНТЕРФЕЙСА МОБИЛЬНОГО ПРИЛОЖЕНИЯ УПРАВЛЕНИЯ ПОКУПКАМИ БИЛЕТОВ

Прохницкий Г.К.
Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь
Осипович Т. А. — кандидат экононических наук

Описано несколько способов оптимизации интерфейса для покупки билетов, нагружаемого большим количеством пользователей и написанного на Laravel.

На начальных этапах разработки приложения мало кто задумывается, что будет на момент очередного обновления или перезапуска приложения. Однако для проектов, которые близятся к своему первому выходу в реальную работу или выходят на новый уровень по количеству использований данные методики оптимизации будут очень полезными.

1. Желательная загрузка: Общей проблемой при извлечении отношений Eloqent ORM является проблема запроса N + 1. Вот базовый сценарий: есть две модели: заказ и пользователь, с отношениями друг к другу. Необходимо получить все заказы и их соответствующих пользователей.

Этот цикл будет выполнять один запрос, чтобы получить все заказы в базе данных, а затем еще один запрос для каждого заказа, чтобы получить пользователя. Чтобы подробнее понять, можно обратиться к примеру:

```
$orders = App\Order::all();
foreach ($orders as $order) {
  echo $order->user->name;
}
```

Для примера: есть 100 заказов, этот цикл будет выполнять 101 запрос: один для оригинального заказа, и еще 100 запросов для получения пользователя каждого заказа. Это может показаться небольшой нагрузкой, потому что в примере небольшой набор данных. При увеличении набора данных увеличится и количество запросов, которые будут делаться по базе данных. Чтобы решить эту проблему, используется жадная загрузка. В момент извлечения всех заказов, в отношении пользователей к заказам будут загружены и владельцы заказов. Таким образом, операция сократится до двух запросов к базе данных.

```
$cars = App\Order::with('user')->get();
foreach ($orders as $order) {
   echo $order->user->name;
}
```

2. Результаты базы данных по частям: Еще одним общим узким местом в производительности при работе с базой данных результатов может быть использование слишком большого объема памяти. Есть загружать тысячи моделей, каждая со своими данными и отношениями, и можно увидеть, как это может привести к ужасной "ошибке допустимого объема памяти, исчерпавшей себя".

Чтобы решить эту проблему необходимо использовать метод Eloquent chunk, который предназначен для сохранения памяти при работе с большими наборами данных.

С новыми группами пользователей по 100 за раз, использование памяти должно оставаться последовательным.

```
User::with('orders')->chunk(100, function($users) {
   foreach ($users as $user) {
      $tasks = $user->orders;
   }
});
```

Стоит обратить внимание, как использование памяти остается стабильным на одном уровне. Теперь неважно, сколько пользователей в БД, эта команда никогда не должна заканчиваться в памяти.

3. Кэширование маршрутов. Кэширование маршрутов резко сократит время, необходимое для регистрации всех маршрутов веб сервиса. Для создания кэша маршрутов используется команда Artisan "route:cache"

Теперь, вместо загрузки маршрутов из корневой папки проекта, файл кэшированных маршрутов будет загружаться при каждом запросе. Поскольку при добавлении новых маршрутов нужно будет генерировать свежий кэш маршрутов, команда "route:cache" должна выполняться только во время установки сервиса на сервере. Для очистки кэша маршрутов можно использовать команду "route:clear". Важно помнить, чтобы использовать кэширование маршрутов, необходимо преобразовать все маршруты в классы контроллеров.

4. Кэширование конфигурации. Так же, как и в случае кэширования маршрутов, можно кэшировать конфигурационные файлы. Используется команда Artisan "config:cache", которая объединит все параметры конфигурации приложения в один файл, который будет быстро загружен фреймворком.

Как и в случае с командой "route:cache", эта команда должна быть запущена только во время развертывания проекта на сервере. Команду не следует запускать во время локальной разработки, так как опции конфигурации часто необходимо менять во время разработки приложения. Если выполнить команду "config:cache" во время установки, нужно быть точно знать, что все переменные, используемые в приложении вызываются из конфигурационных файлов настроек приложения. После того, как конфигурация будет кэширована, файл временных настроек ".env" не будет загружен, и все вызовы функции "env" вернутся к нулю.

5. Очереди. Очереди также могут быть использованы для улучшения производительности приложения. Для примера есть приложение, которое при регистрации отправляет приветственное письмо новым пользователям. Когда пользователь заполнит и отправит форму регистрации, в базу данных будет вставлена новая пользовательская запись, после чего сервис почтовой рассылки сделает запрос для отправки электронного письма.

Фактическая отправка письма может занять несколько секунд (или, в зависимости от обстоятельств, миллисекунд). Чтобы не заставлять пользователей думать, что процесс регистрации медленный, можно использовать очередь, поставив отправку сообщения в очередь, чтобы запустить его в качестве фоновой задачи. Таким образом, наши пользователи получат быстрый ответ из формы регистрации, в то время как процесс отправки электронной почты будет выполняться в фоновом режиме.

6. Кэширование базы данных. С помощью приложений кэширования данных, таких как Redis и Memcached, можно кэшировать результаты базы данных. Вместо того, чтобы снова и снова извлекать один и тот же набор результатов из базы данных, можно использовать кэширование базы данных, сначала извлекая записи из базы данных, а затем кэшируя их для последующего использования.

Все вышеперечисленные примеры используются в реальных условиях работы при разработке интерфейса мобильного приложения управления покупками билетов. Так же они могут быть применены и к другим проектам схожей структуры, для разработки которых используется фреймворк Laravel.

Список использованных источников:

1. ***Easy E-Commerce Using Laravel and Stripe*** */ W. Jason Gilmore and Eric L. Barnes // easycommercebook.com, 2015. – 110c.*

2. Design Patterns in PHP and Laravel // Kelt Dockins //Apress, 2017. – 238c.