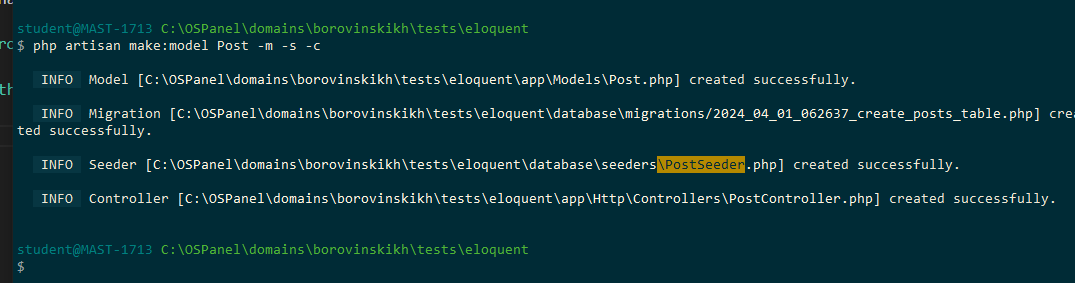
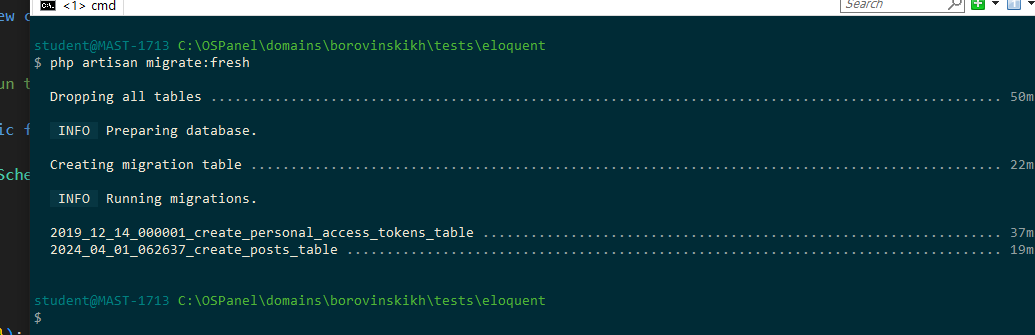
**Практическая работа Модели в Laravel**

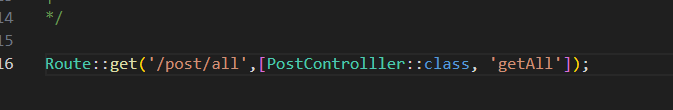
1. Сделайте через PhpMyAdmin таблицу **posts** (статьи) с полями **id**, **title** (заголовок), **desc** (короткое описание статьи), **text** (текст статьи), **date** (дата публикации). Добавьте в эту таблицу 5 статей.
2. Сделайте модель Eloquent для таблицы **posts** из предыдущей задачи.
3. Сделайте контроллер **PostController** для работы со статьями.

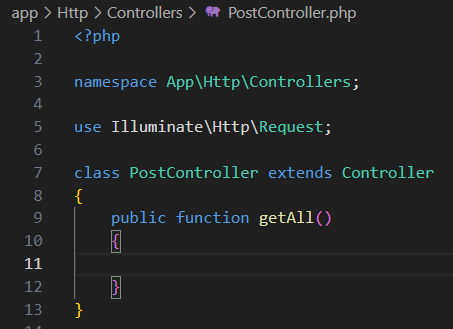






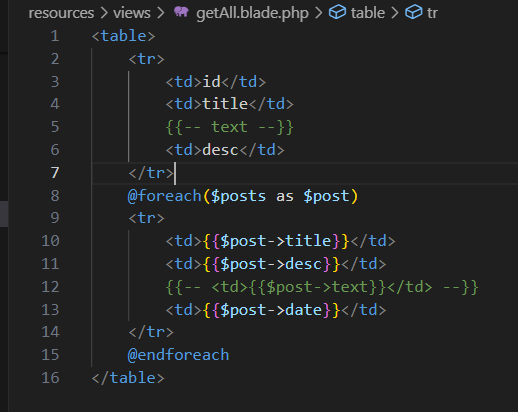
1. В контроллере **PostController** сделайте действие **getAll** для получения списка всех статей. Пусть это действие будет доступно по адресу **/post/all/**.

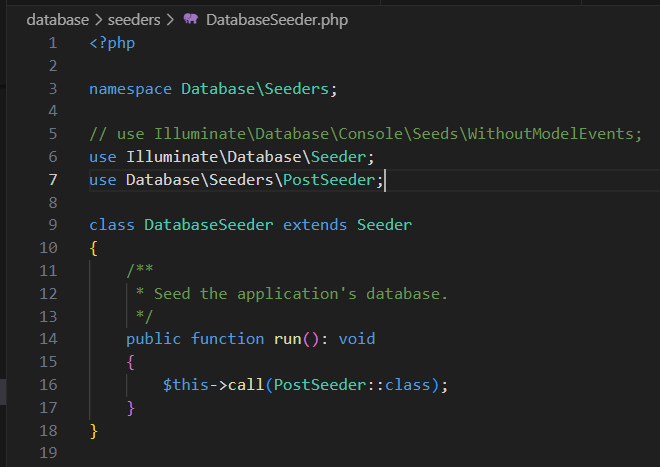


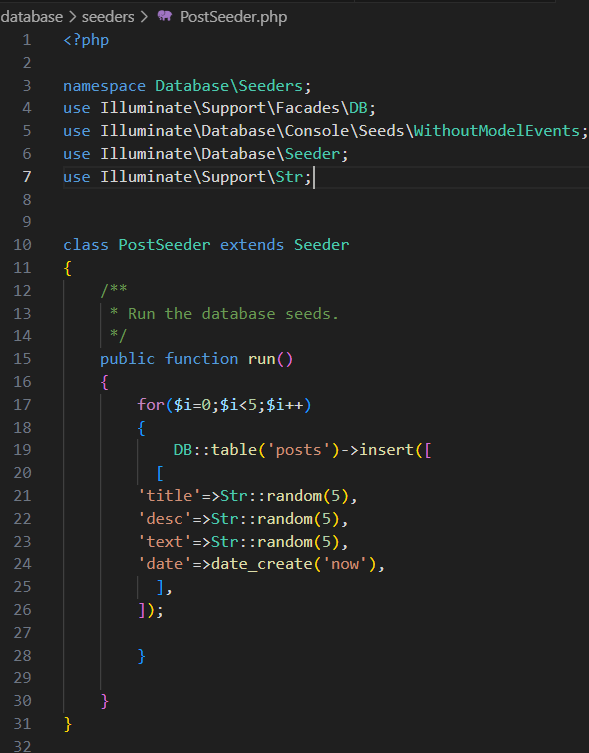


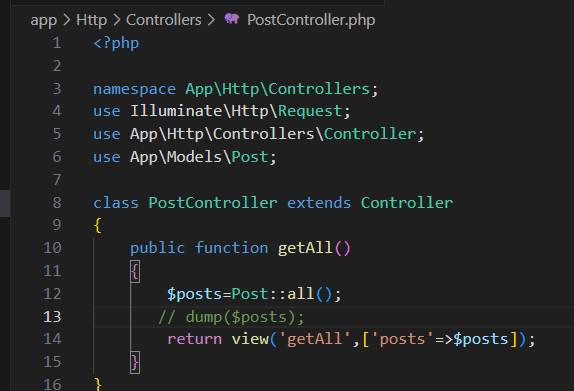
1. Используя модель **Post**, созданную в предыдущем уроке для таблицы **posts**, получите в действии **getAll** массив всех статей, передайте его в представление и выведите циклом на экран в виде HTML таблицы с колонками **id**, **title** (заголовок) и **desc** (то есть без текста статьи).



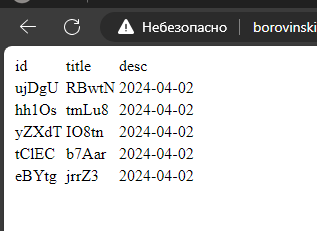






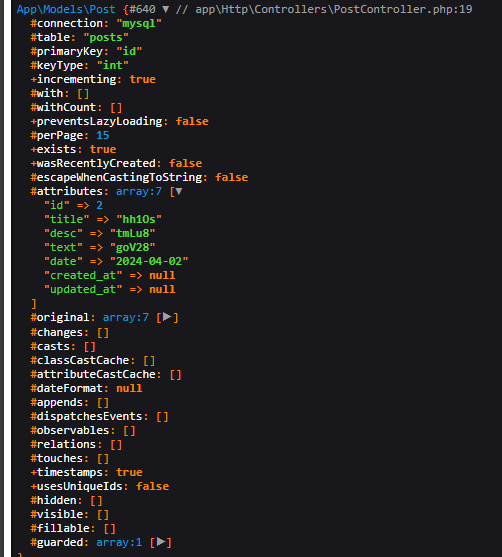


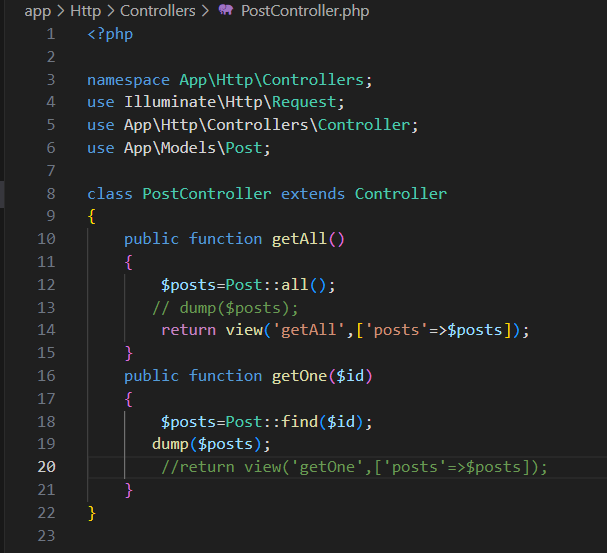




1. В контроллере **PostController** сделайте действие **getOne** для получения одной статьи по ее id. Пусть это действие будет доступно по адресу **/post/:id/**, где **:id** представляет собой **id** желаемой записи.Не забудьте наложить ограничение регулярным выражением на наш параметр.



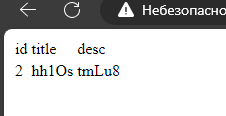




1. В действии **getOne** из таблицы **posts** получите статью соответствующую переданному параметру. Передайте полученную статью в представление и выведите ее на экран, оформив соответствующим HTML кодом.

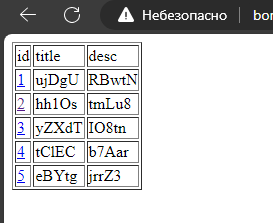


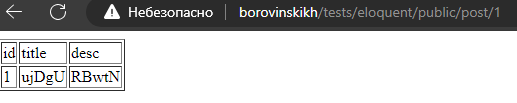




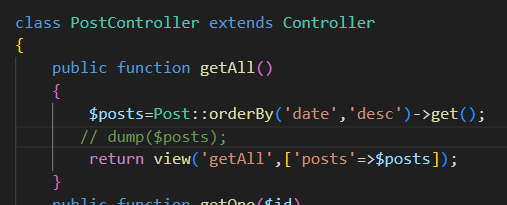
1. Отредактируйте представление действия **getAll** так, чтобы заголовок каждой из выводимых статей стал ссылкой на соответствующую страницу одной статьи.



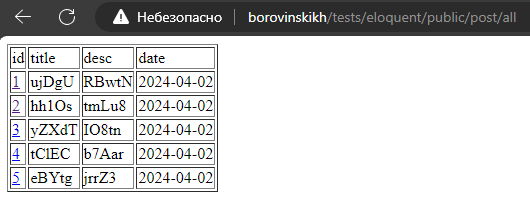




1. Отредактируйте действие **getAll** так, чтобы получаемые статьи были отсортированы по убыванию даты публикации.

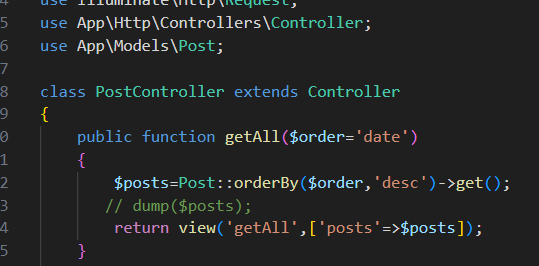


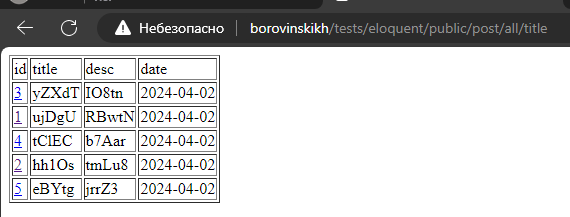


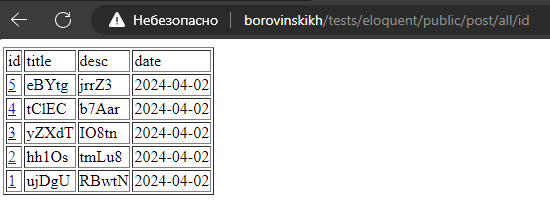


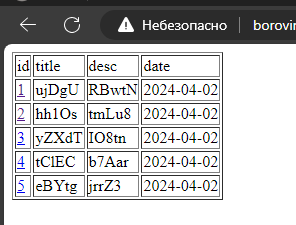
1. Отредактируйте маршрут действия **getAll** так, чтобы вместо адреса **/post/all/** нашей действие стало доступно по адресу **/post/all/:order/**, где **:order** представляет собой имя поля, по которому выполнять сортировку.Сделайте так, чтобы сортировать можно было по полям **id**, **title** и **date**. Причем сортировка должна быть по убыванию значения поля.Сделайте так, чтобы параметр **:order** был не обязательным и по умолчанию имел значение **date**.





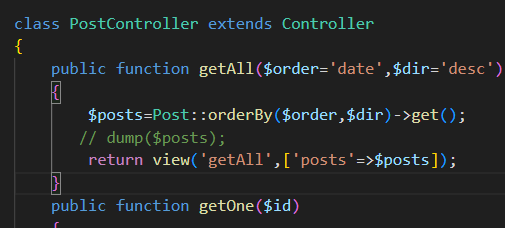


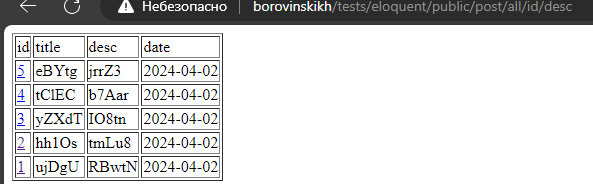


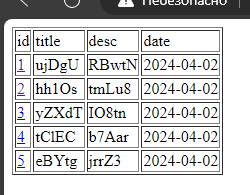


1. Отредактируйте маршрут действия **getAll** так, чтобы появился еще один параметр **:dir**, представляющий собой направление сортировки (по убыванию или по возрастанию). То есть наш маршрут станет выглядеть так: **/post/all/:order/:dir**.Пусть параметр **:dir** может иметь только два значения: **asc** или **desc**. При этом пусть наш параметр также является не обязательным и по умолчанию имеет значение **desc**.

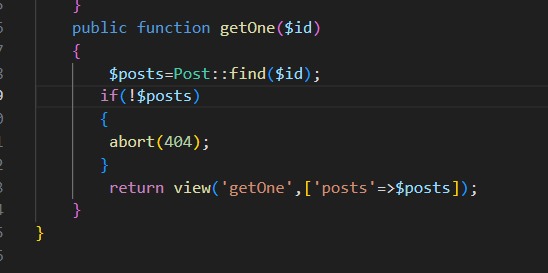


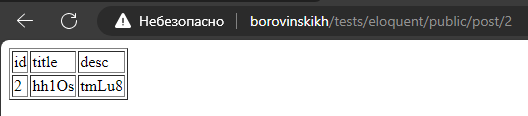


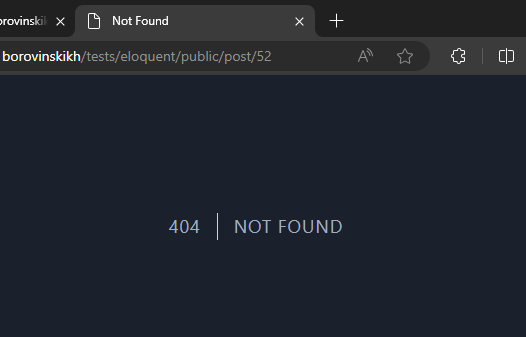




1. Отредактируйте действия **getOne** так, чтобы, если для переданного параметром id не существует записи в базе данных, то выводилась ошибка 404.

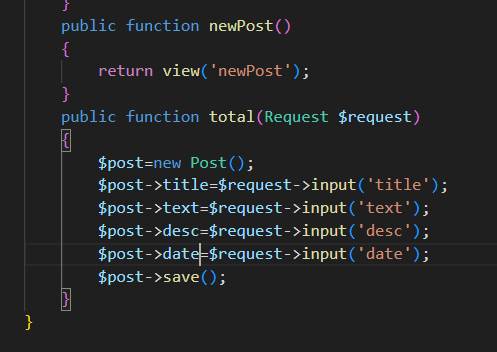


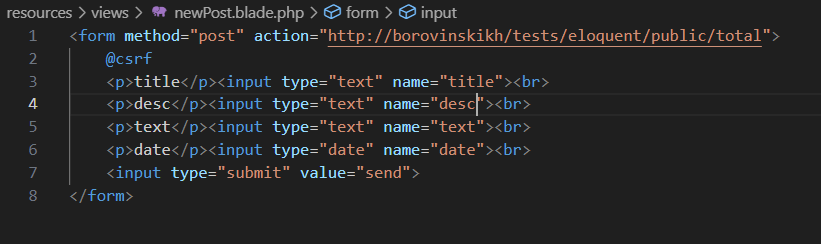


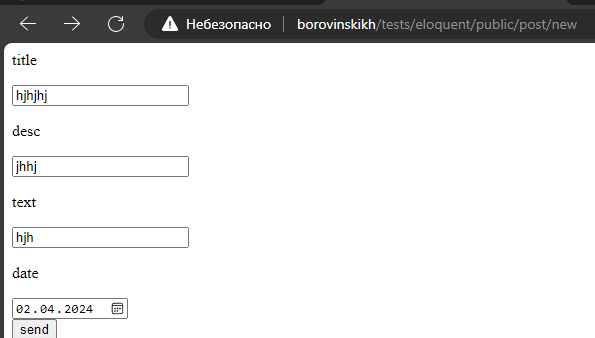


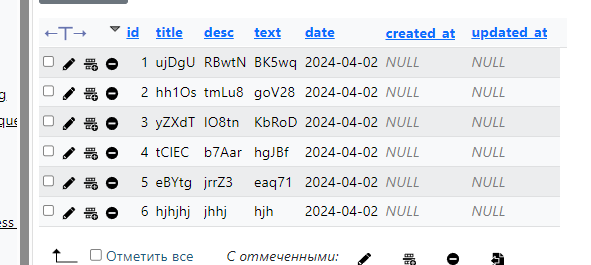
1. В контроллере **PostController**, созданном в предыдущем уроке, сделайте действие **newPost** для создания новой статьи. Пусть это действие будет доступно по адресу **/post/new/**. В представлении действия покажите форму для добавления новой записи. После отправки формы сохраните новую запись.











1. Сделайте какое-нибудь действие, по заходу на которое для статьи с id 1 поменяйте **title** и короткое описание **desc**.
2. Давайте теперь сделаем отдельное действие для редактирования статьи. Сделаем так, чтобы при заходе на это действие показывалась форма, уже заполненная данными из базы данных.При желании эти данные можно будет поменять и обновить их в базе данных.Итак, сделаем в контроллере **PostController** действие **editPost**:

**namespace** App\Http\Controllers;

**use** App\Http\Controllers\Controller;

**class** PostController **extends** Controller

{

**public function** editPost()

   {

**return** view(**'test.editPost'**);

   }

}

В представлении действия покажем форму для редактирования статьи:

<**form** action=**""** method=**"POST"**>

  <**input** name=**"title"**>

  <**input** name=**"desc"**>

  <**input** name=**"date"**>

  <**textarea** name=**"text"**></**textarea**>

  <**input** name=**"submit"** type=**"submit"**>

</**form**>

Так как форма отправляется методом **POST**, то для маршрута, соответствующего нашему действию, разрешим как метод **get**, так и **post**:

Route::match([**'get'**, **'post'**], **'post/edit/{id}'**, **'PostController@editPost'**);

Обратите внимание на то, что в адресе маршрута записан параметр **id**, представляющий собой **id** записи, редактирование которой мы осуществляем.

Давайте доработаем наш контроллер, подключив нужные классы и выполнив инъекцию класса Request, а также передав параметр **id** в метод действия **editPost**:

**namespace** App\Http\Controllers;

**use** App\Http\Controllers\Controller;

**use** Illuminate\Http\Request;

// подключаем класс Request

**use** App\Post;

// подключаем модель Post

**class** PostController **extends** Controller

{

**public function** editPost(Request $request, $id)

   {

**return** view(**'test.editPost'**);

   }

}

Давайте теперь сделаем так, чтобы по заходу на страницу данные записи с переданным **id** уже были записаны в форме.

Для этого получим запись по ее **id** и передадим полученные данные в представление:

**namespace** App\Http\Controllers;

**use** App\Http\Controllers\Controller;

**use** Illuminate\Http\Request;

// подключаем класс Request

**use** App\Post;

// подключаем модель Post

**class** PostController **extends** Controller

{

**public function** editPost(Request $request, $id)

   {

       $post = Post::find($id); // получаем запись по id

**return** view(**'test.editPost'**, [**'post'** => $post]);

   }

}

Выведем переданные данные в представлении:

<**form** action=**""** method=**"POST"**>

  <**input** name=**"title"** value=**"{{** $post->**title }}"**>

  <**input** name=**"desc"** value=**"{{** $post->**desc }}"**>

  <**input** name=**"date"** value=**"{{** $post->**date }}"**>

  <**textarea** name=**"text"**>{{ $post->**text** }}</**textarea**>

  <**input** name=**"submit"** type=**"submit"**>

</**form**>

Давайте теперь поймаем отправку формы. Здесь я использую хитрость: не буду проверять наличие всех полей формы, а просто проверю с помощью метода **has** наличие в **$request** поля **submit**. Данное имя соответствует кнопке отправки формы (я дал этой кнопке атрибут name в значении **submit**).

Итак, вот проверка:

**namespace** App\Http\Controllers;

**use** App\Http\Controllers\Controller;

**use** Illuminate\Http\Request;

**use** App\Post;

**class** PostController **extends** Controller

{

**public function** editPost(Request $request, $id)

   {

       $post = Post::find($id);

       // Если форма была отправлена...

**if** ($request->has(**'submit'**)) {

       }

**return** view(**'test.editPost'**, [**'post'** => $post]);

   }

}

Давайте теперь по отправке формы запишем данные формы в нашу модель и сохраним ее:

**namespace** App\Http\Controllers;

**use** App\Http\Controllers\Controller;

**use** Illuminate\Http\Request;

**use** App\Post;

**class** PostController **extends** Controller

{

**public function** editPost(Request $request, $id)

   {

       $post = Post::find($id);

       // Если форма была отправлена...

**if** ($request->has(**'submit'**)) {

           $post->**title** = $request->**title**;

           $post->**desc** = $request->**desc**;

           $post->**date** = $request->**date**;

           $post->**text** = $request->**text**;

           $post->save(); // сохраним модель

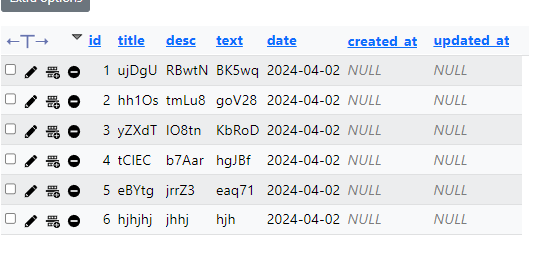
       }

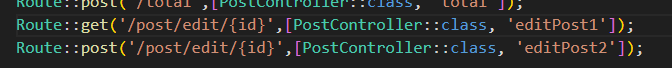
**return** view(**'test.editPost'**, [**'post'** => $post]);

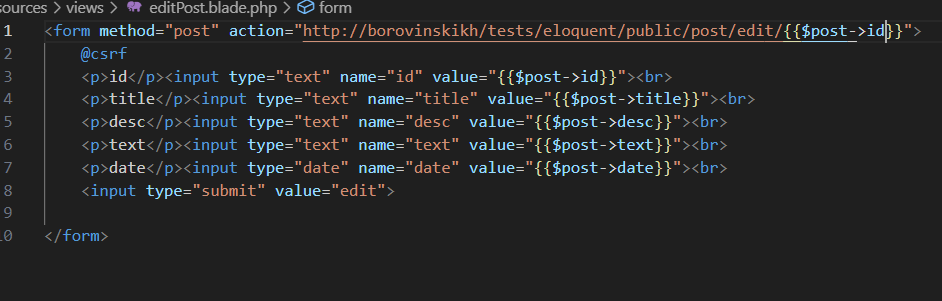
   }

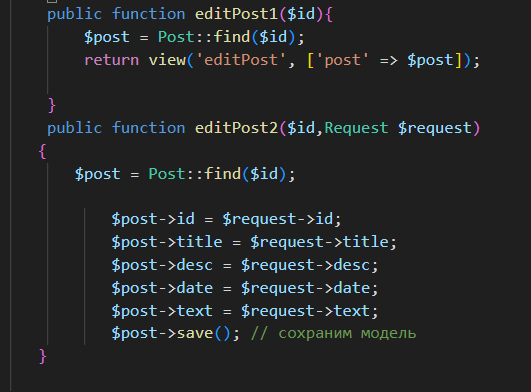
}

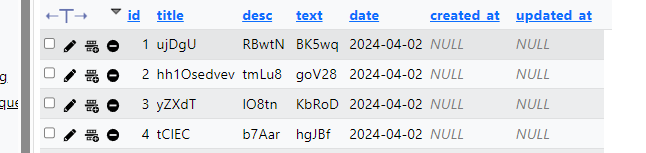
Обратите внимание на то, что после попадания в иф, поля нашей модели **$post** изменятся и уже в измененном виде попадут в представление - то есть в форме после ее отправки уже будут стоять не изначальные значения из БД, а те, которые вбил пользователь. Вот такой хитрый прием.



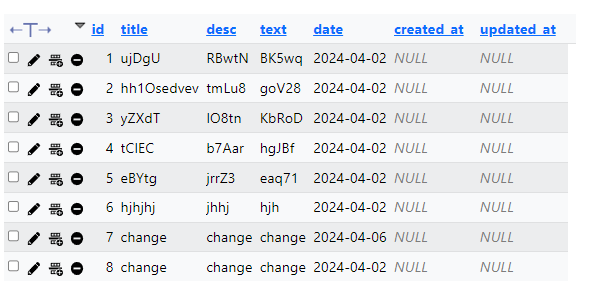


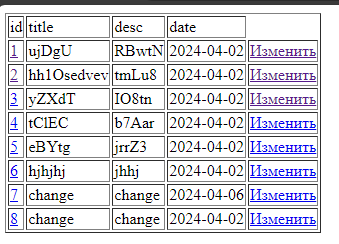


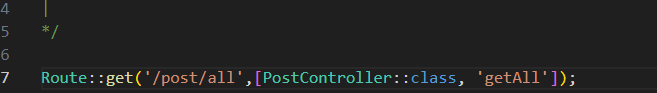


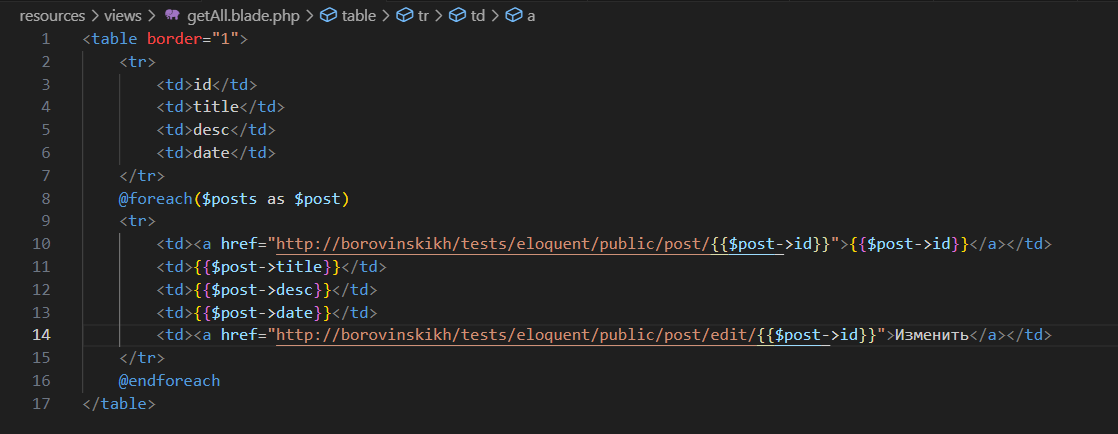


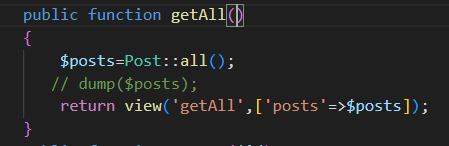
1. Отредактируйте представление действия **getAll** так, чтобы появилась еще одна колонка со ссылкой на редактирование соответствующей статьи.

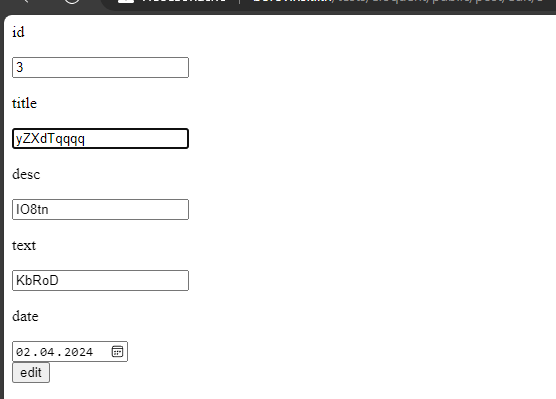


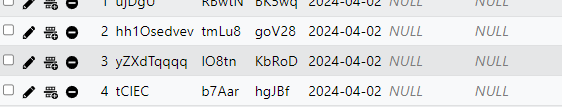




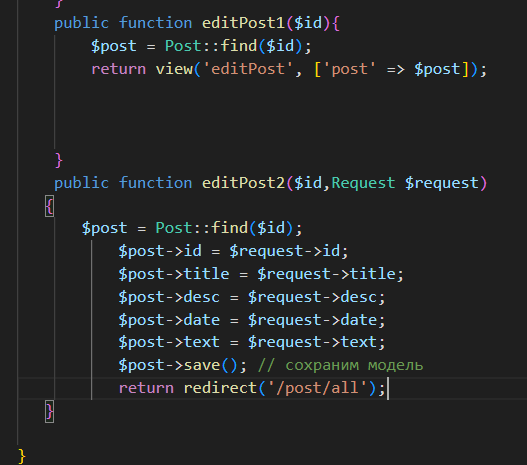




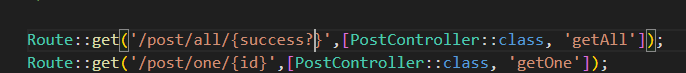


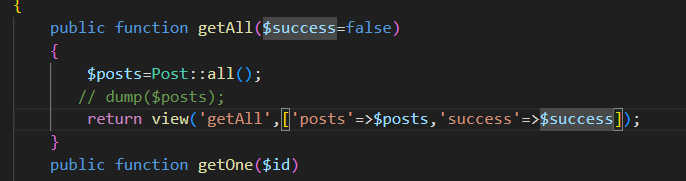


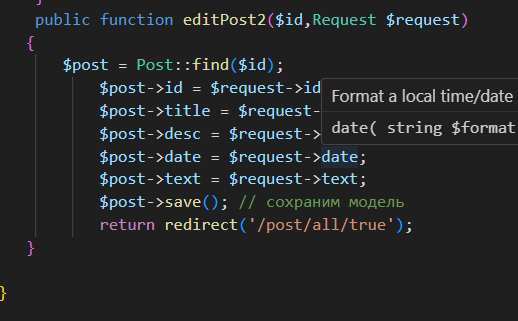
1. Модифицируйте код действия **editPost** так, чтобы после сохранения формы выполнялся редирект на список всех записей (то есть на действие **getAll**).

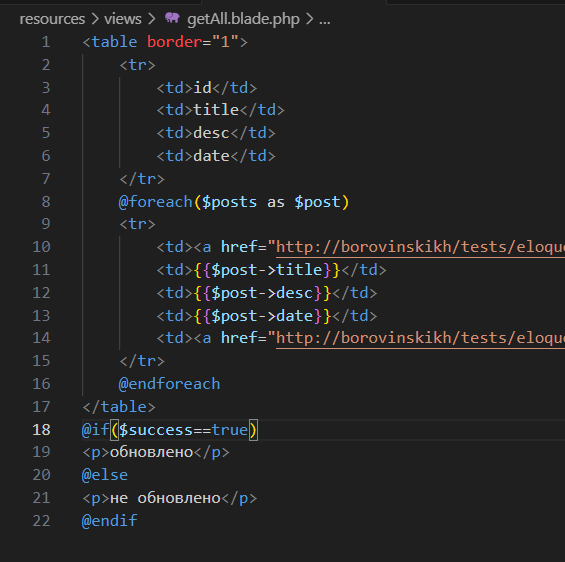


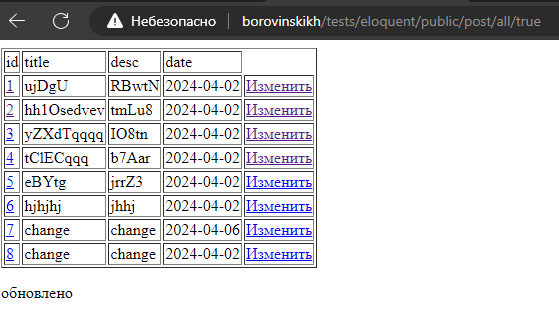
1. Модифицируйте предыдущую задачу так, чтобы при редиректе отправлялось флеш сообщение об успешном обновлении записи. Выводите это сообщение в представлении действия **getAll**.

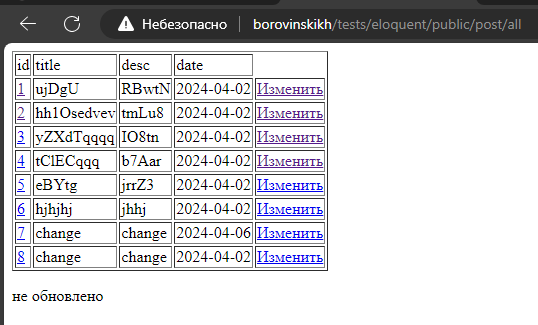






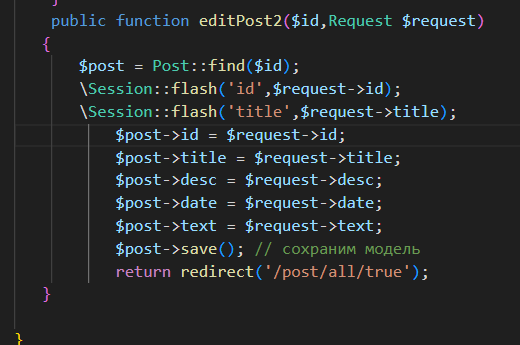


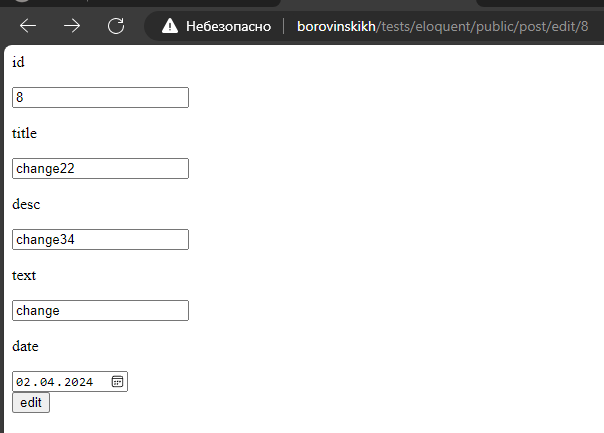




1. Модифицируйте предыдущую задачу так, чтобы во флеш сообщении был указан id и title статьи, подвергнувшейся изменению.

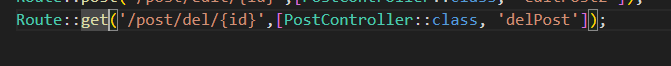




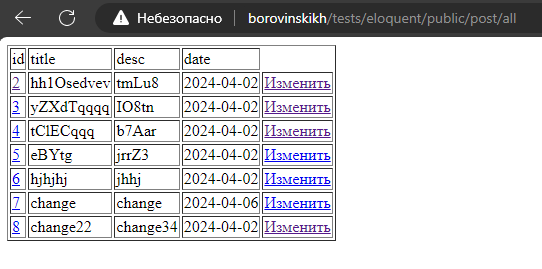




20. В контроллере **PostController**, созданном в предыдущих уроках, сделайте действие **delPost** для удаления статьи. Пусть это действие будет доступно по адресу **/post/del/:id**, где **:id** будет содержать **id** статьи для удаления.



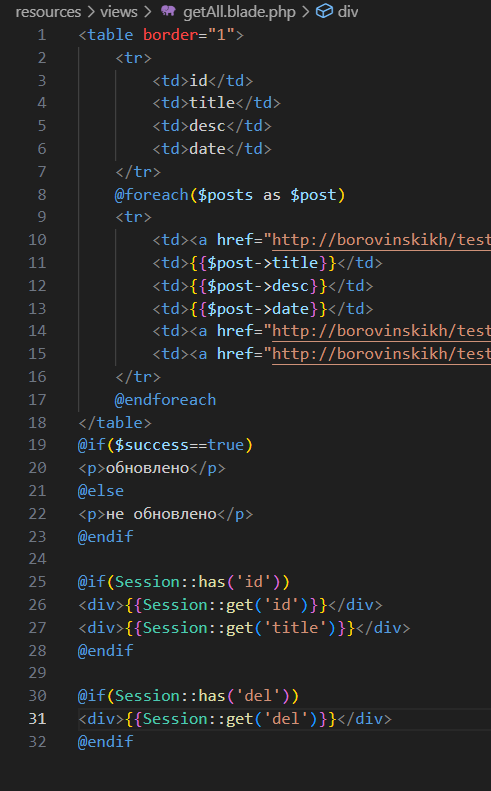


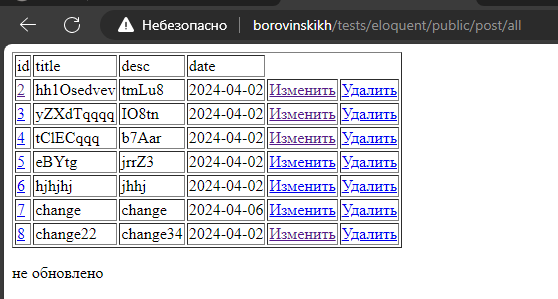


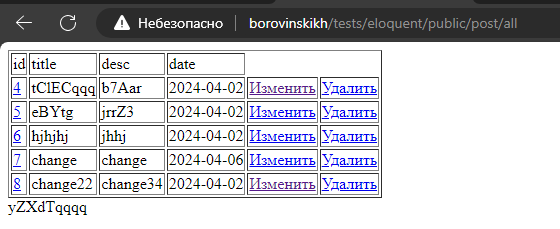
21. Отредактируйте представление действия **getAll** так, чтобы появилась еще одна колонка со ссылкой на удаления соответствующей статьи.

После удаления выполняйте редирект обратно на список статей с флеш сообщением об успешном удалении статьи. Сообщение должно содержать **title** удаленной статьи.

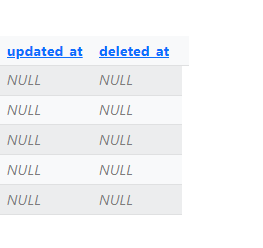


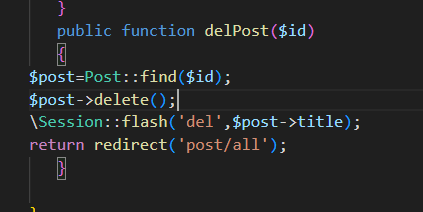


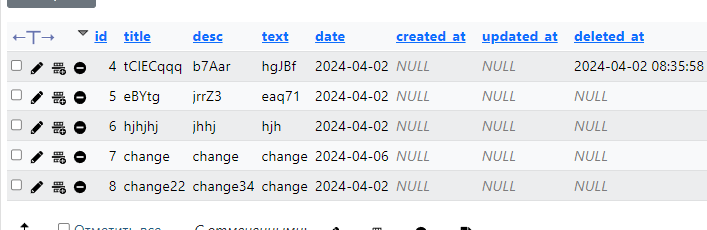


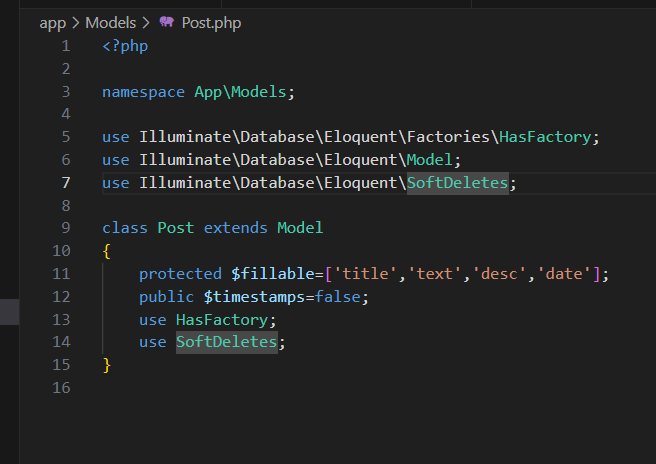


22. Реализуйте мягкое удаление статей.

Мягкое удаление означает, что запись на самом деле остаётся в базе данных, но в таблице для записи устанавливается поле deleted\_at. 

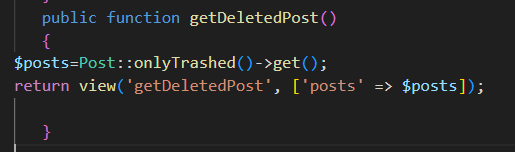


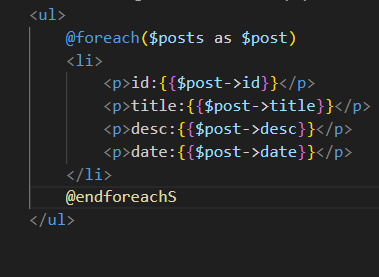


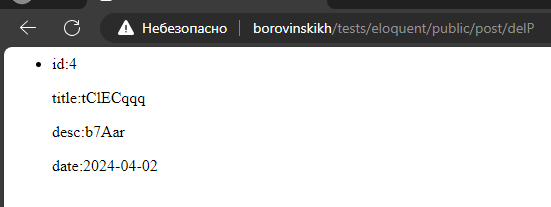


23. Сделайте действие **getDeletedPost**, выводящее список удаленных статей.

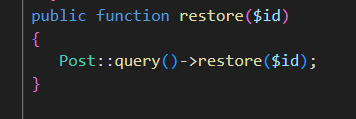








24. Сделайте действие **restorePost**, восстанавливающее удаленную статью обратно.



25. В представлении действия **getDeletedPost** для каждой статьи сделайте ссылку на ее восстановление.



