### **Заметки к занятию 2. 05.11.2024**

В этом занятии мы сосредоточимся на работе с отношениями hasOne и hasMany, политиками (policies) и ресурсами (resources) в Laravel.

#### **1. Работа с отношениями hasOne и hasMany**

Laravel предоставляет удобные методы для работы с отношениями между моделями. Вот основные примеры использования.

**Связь hasOne**:

Отношение «один к одному» (hasOne) используется, когда одна запись в таблице может быть связана с одной записью в другой таблице.

Например, если у нас есть таблица users и таблица profiles, где каждому пользователю соответствует один профиль, мы можем определить отношение hasOne в модели User:  
  
class User extends Model

{

public function profile()

{

return $this->hasOne(Profile::class);

}

}

**Связь hasMany**:

Отношение «один ко многим» (hasMany) используется, когда одна запись в таблице может иметь множество связанных записей в другой таблице.

Например, у каждого пользователя может быть много статей (articles), тогда в модели User мы можем определить hasMany:  
  
class User extends Model

{

public function articles()

{

return $this->hasMany(Article::class);

}

}

#### **2. Создание миграций**

Миграции позволяют определять и изменять структуру базы данных с помощью кода, что облегчает управление изменениями в базе данных.

**Создание новой миграции**:

Команда для создания новой миграции, например, для таблицы gagarin\_flights:  
  
php artisan make:migration create\_gagarin\_flights\_table

Это создаст файл в папке database/migrations, где можно определить структуру таблицы gagarin\_flights.

#### **3. Ресурсы (Resources)**

Ресурсы Laravel позволяют форматировать данные перед отправкой их в API или в клиентское приложение, благодаря чему можно легко контролировать структуру ответа.

**Создание ресурса**:

Чтобы создать ресурс LunarMissionResource, выполните команду:  
  
php artisan make:resource LunarMissionResource

Ресурс создается в папке app/Http/Resources. Он позволяет настроить, какие данные и в каком виде будут возвращаться клиенту.

Пример использования в ресурсе:  
  
class LunarMissionResource extends JsonResource

{

public function toArray($request)

{

return [

'id' => $this->id,

'name' => $this->name,

'launch\_date' => $this->launch\_date,

'status' => $this->status,

];

}

}

#### **4. Политики (Policies)**

Политики Laravel (policies) позволяют контролировать доступ к определенным действиям в приложении, например, разрешить пользователю редактировать или удалять ресурсы.

**Создание политики**:

Политики обычно создаются для моделей. Например, чтобы создать политику для модели Post, выполните команду:  
  
php artisan make:policy PostPolicy

Политика будет создана в папке app/Policies, и в ней можно определить методы для различных разрешений (например, update, delete, view).

Пример метода в политике PostPolicy:  
  
class PostPolicy

{

public function update(User $user, Post $post)

{

return $user->id === $post->user\_id;

}

}

Здесь проверяется, что только автор поста может его обновить.

**Применение политики**:

После создания политики нужно зарегистрировать ее в AuthServiceProvider, чтобы Laravel знал, какую политику применять к модели:  
  
protected $policies = [

Post::class => PostPolicy::class,

];

#### **5. Работа с Postman**

При работе с API можно использовать Postman для отправки запросов и тестирования ответов. Если для авторизации используется токен, его можно сохранить в переменной окружения для последующих запросов:

pm.environment.set("token", pm.response.json().token);

Это позволяет автоматизировать передачу токена в заголовке запроса, делая процесс тестирования API более удобным.