Реализация API сервера и AJAX запросов

Шаг 1: Установка Laravel

composer create-project --prefer-dist laravel/laravel имя-проекта

cd имя-проекта

Шаг 2: Создание миграций и моделей

php artisan make:migration create\_posts\_table

php artisan make:migration create\_comments\_table

Миграции

create\_posts\_table.php:

<?

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

public function up()

{

    Schema::create('posts', function (Blueprint $table) {

        $table->id();

        $table->string('title');

        $table->text('content');

        $table->timestamps();

    });

}

create\_comments\_table.php

<?

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

public function up()

{

    Schema::create('comments', function (Blueprint $table) {

        $table->id();

        $table->unsignedBigInteger('post\_id');

        $table->text('comment\_text');

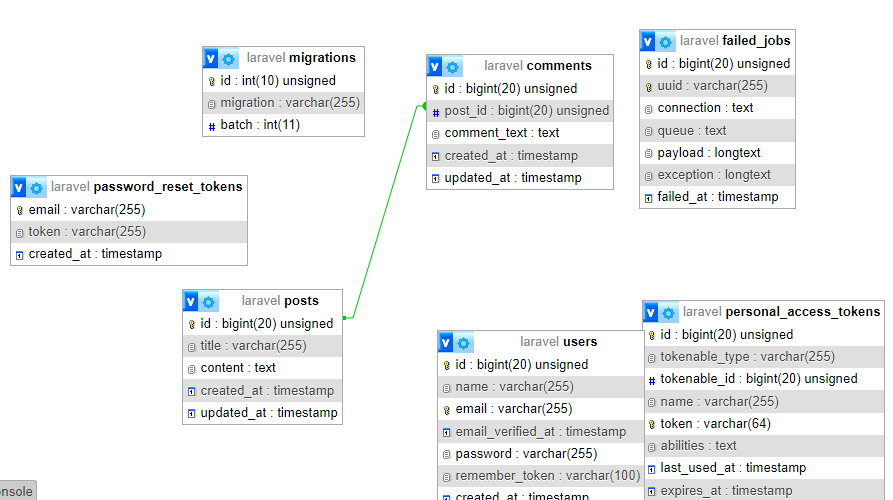
        $table->timestamps();

        $table->foreign('post\_id')->references('id')->on('posts')->onDelete('cascade');

    });

}

Обмеження ON DELETE CASCADE використовується в MySQL для автоматичного видалення рядків із дочірньої таблиці, коли видаляються рядки з батьківської таблиці. Наприклад, коли студент реєструється на платформі онлайн-навчання, усі дані про студента записуються з його унікальним номером/ідентифікатором.



Шаг 3: Выполнение миграций

php artisan migrate

Шаг 4: Создание моделей

php artisan make:model Post

php artisan make:model Comment

Post.php

<?

namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model

{

    protected $fillable = ['title', 'content'];

    public function comments()

    {

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

    }

}

Comment.php

<?

namespace App;

use Illuminate\Database\Eloquent\Model;

class Comment extends Model

{

    protected $fillable = ['post\_id', 'comment\_text'];

    public function post()

    {

        return $this->belongsTo(Post::class);

    }

}

Шаг 5: Создание контроллера для API

php artisan make:controller ApiController

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Post;

class ApiController extends Controller

{

    public function index()

    {

        $posts = Post::with('comments')->get();

        return response()->json($posts);

    }

    public function store(Request $request)

    {

        $post = Post::create([

            'title' => $request->input('title'),

            'content' => $request->input('content')

        ]);

        return response()->json($post);

    }

public function show($id)

    {

        $post = Post::with('comments')->find($id);

        return response()->json($post);

    }

    public function update(Request $request, $id)

    {

        $post = Post::find($id);

        $post->update([

            'title' => $request->input('title'),

            'content' => $request->input('content')

        ]);

        return response()->json($post);

    }

public function destroy($id)

    {

        Post::destroy($id);

        return response()->json(['message' => 'Пост удален']);

    }

}

CommentController

php artisan make:controller CommentController

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Comment;

class CommentController extends Controller

{

public function store(Request $request, $postId)

    {

        $comment = Comment::create([

            'post\_id' => $postId,

            'comment\_text' => $request->input('comment\_text')

        ]);

        return response()->json($comment);

    }

    public function index($postId)

    {

        $comments = Comment::where('post\_id', $postId)->get();

        return response()->json($comments);

    }

    public function update(Request $request, $commentId)

    {

        $comment = Comment::find($commentId);

        $comment->update([

            'comment\_text' => $request->input('comment\_text')

        ]);

        return response()->json($comment);

    }

    public function destroy($commentId)

    {

        Comment::destroy($commentId);

        return response()->json(['message' => 'Комментарий удален']);

    }

}

Шаг 6: Определение маршрутов API

php artisan install:api

routes/api.php

<?php

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\ApiController;

use App\Http\Controllers\CommentController;

// Маршрут для получения списка всех постов

Route::get('/posts', [ApiController::class, 'index']);

// ApiController::class - указывает на контроллер ApiController

// 'index' - метод контроллера, который будет вызываться при запросе

// Маршрут для создания нового поста

Route::post('/posts', [ApiController::class, 'store']);

// Маршрут для получения информации о конкретном посте

Route::get('/posts/{id}', [ApiController::class, 'show']);

// {id} - параметр в URL, который передается в метод show

// Маршрут для обновления информации о посте

Route::put('/posts/{id}', [ApiController::class, 'update']);

// Маршрут для удаления поста

Route::delete('/posts/{id}', [ApiController::class, 'destroy']);

// Маршрут для создания комментария к посту

Route::post('/posts/{id}/comments', [CommentController::class, 'store']);

// Маршрут для получения комментариев к конкретному посту

Route::get('/posts/{id}/comments', [CommentController::class, 'index']);

// Маршрут для обновления информации о комментарии

Route::put('/comments/{commentId}', [CommentController::class, 'update']);

// Маршрут для удаления комментария

Route::delete('/comments/{commentId}', [CommentController::class, 'destroy']);

Шаг 7: Реализация CRUD-запросов на AJAX

index.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Laravel CRUD API</title>

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

<body>

    <header>

        <nav class="navbar navbar-expand-lg navbar-light bg-light">

            <a class="navbar-brand" href="#">Laravel CRUD</a>

        </nav>

    </header>

<div class="container">

        <h1>Посты</h1>

        <form id="post-form">

            <div class="form-group">

                <label for="title">Заголовок</label>

                <input type="text" class="form-control" id="title" name="title" required>

            </div>

            <div class="form-group">

                <label for="content">Содержание</label>

                <textarea class="form-control" id="content" name="content" required></textarea>

            </div>

            <button type="submit" class="btn btn-primary">Создать пост</button>

        </form>

        <table class="table">

            <thead>

                <tr>

                    <th>ID</th>

                    <th>Заголовок</th>

                    <th>Содержание</th>

                    <th>Действия</th>

                </tr>

            </thead>

            <tbody id="post-list">

                <!-- Сюда будут добавляться посты с помощью JavaScript -->

            </tbody>

        </table>

    </div>

    <footer>

        <div class="container">

            <p>&copy; 2023 Laravel CRUD API</p>

        </div>

    </footer>

    <script src="script.js"></script>

</body>

</html>

edit.html

<!-- edit.html -->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Laravel CRUD API - Редактировать пост</title>

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

<body>

    <header>

        <nav class="navbar navbar-expand-lg navbar-light bg-light">

            <a class="navbar-brand" href="index.html">Laravel CRUD</a>

        </nav>

    </header>

    <div class="container">

        <h1>Редактировать пост</h1>

        <form id="post-form">

            <div class="form-group">

                <label for="title">Заголовок</label>

                <input type="text" class="form-control" id="title" name="title" required>

            </div>

            <div class="form-group">

                <label for="content">Содержание</label>

                <textarea class="form-control" id="content" name="content" required></textarea>

            </div>

            <button type="submit" class="btn btn-primary">Сохранить изменения</button>

        </form>

    </div>

    <footer>

        <div class="container">

            <p>&copy; 2023 Laravel CRUD API</p>

        </div>

    </footer>

    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

    <script src="script.js"></script>

</body>

</html>

create.html

<!-- create.html -->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Laravel CRUD API - Создать пост</title>

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

<body>

    <header>

        <nav class="navbar navbar-expand-lg navbar-light bg-light">

            <a class="navbar-brand" href="index.html">Laravel CRUD</a>

        </nav>

    </header>

    <div class="container">

        <h1>Создать пост</h1>

        <form id="post-form">

            <div class="form-group">

                <label for="title">Заголовок</label>

                <input type="text" class="form-control" id="title" name="title" required>

            </div>

            <div class="form-group">

                <label for="content">Содержание</label>

                <textarea class="form-control" id="content" name="content" required></textarea>

            </div>

            <button type="submit" class="btn btn-primary">Создать пост</button>

        </form>

    </div>

    <footer>

        <div class="container">

            <p>&copy; 2023 Laravel CRUD API</p>

        </div>

    </footer>

    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

    <script src="script.js"></script>

</body>

</html>

script.js

document.addEventListener('DOMContentLoaded', () => {

    // Загрузка списка постов при загрузке страницы

    loadPosts();

});

// Загрузка списка постов

function createPost() {

    // URL API, до якого ви плануєте звертатися

const apiUrl = 'http://localhost:8000/api/posts';

const title = document.getElementById('titleId').value;

const content = document.getElementById('contentId').value;

// Дані post у форматі URLSearchParams

const productData = new URLSearchParams();

productData.append('title', title);

productData.append('content', content);

// Опції для POST-запиту

const requestOptions = {

    method: 'POST',

    headers: {

        'Content-Type': 'application/x-www-form-urlencoded'

    },

    body: productData

};

// Відправка POST-запиту

fetch(apiUrl, requestOptions)

    .then(response => {

        if (response.ok) {

            console.log('Post успішно додано!');

        } else {

            console.error('Помилка під час додавання post. Статус код:', response.status);

        }

    })

    .catch(error => {

        console.error('Виникла помилка:', error);

    });

}

// Загрузка списка постов

function loadPosts() {

    const apiUrl = 'http://localhost:8000/api/posts';

    var xhr = new XMLHttpRequest();

    xhr.open('GET', apiUrl, true);

    xhr.onreadystatechange = function () {

        if (xhr.readyState === 4) {

            if (xhr.status === 200) {

                var data = JSON.parse(xhr.responseText);

                var postList = document.getElementById('post-list');

                postList.innerHTML = '';

                data.forEach(function (post) {

                    var row = document.createElement('tr');

                    row.innerHTML = `

                        <td>${post.id}</td>

                        <td>${post.title}</td>

                        <td>${post.content}</td>

                        <td>

                            <button class="btn btn-info" onclick="editPost(${post.id})">Редактировать</button>

                            <button class="btn btn-danger" onclick="deletePost(${post.id})">Удалить</button>

                        </td>

                    `;

                    postList.appendChild(row);

                });

            } else {

                console.log('Ошибка: ' + xhr.status);

            }

        }

    };

    xhr.send();

}

// Удаление поста

    function deletePost(id) {

        var xhr = new XMLHttpRequest();

        xhr.open('DELETE', '/api/posts/' + id, true);

        xhr.onreadystatechange = function () {

            if (xhr.readyState === 4) {

                if (xhr.status === 200) {

                    loadPosts();

                } else {

                    console.log('Ошибка: ' + xhr.status);

                }

            }

        };

        xhr.send();

    }

    // Редактирование поста

    function editPost(id) {

        // Реализуйте логику редактирования поста здесь

    }

});

Добавление Auth и Passport

Auth2

Шаг 1: Установка Passport

Откройте терминал и перейдите в каталог вашего Laravel-проекта. Затем выполните следующую команду для установки Laravel Passport:

composer require laravel/passport

Шаг 2: Публикация конфигурации и миграций

Выполните команду для публикации конфигурации Passport:

php artisan vendor:publish --tag=passport-config

Затем выполните миграции для создания таблиц, необходимых для Passport:

php artisan migrate

Шаг 3: Настройка Passport

Откройте файл config/auth.php и укажите api провайдер аутентификации как passport:

    'guards' => [

        'web' => [

            'driver' => 'session',

            'provider' => 'users',

        ],

        'api' => [

            'driver' => 'passport',

            'provider' => 'users',

        ],

    ],

Шаг 4: Генерация ключей и настройка Passport

Запустите команду для генерации ключей и установки Passport:

php artisan passport:install

Это сгенерирует клиентские и серверные ключи для Passport.

Шаг 5: Обновление модели User

Откройте модель User (обычно находится в app/Models/User.php) и добавьте трейт HasApiTokens:

class User extends Authenticatable

{

    use HasApiTokens

Шаг 6: Настройка маршрутов

В вашем файле routes/api.php добавьте маршруты, связанные с Passport. Вот пример, как это может выглядеть:

use Illuminate\Support\Facades\Route;

Route::post('/login', 'AuthController@login');

Route::post('/register', 'AuthController@register');

Шаг 7: Создание контроллера для аутентификации

Создайте контроллер AuthController с методами для аутентификации (регистрации, входа и выхода). Пример метода register:

    public function register(Request $request)

{

    $validator = Validator::make($request->all(), [

        'name' => 'required',

        'email' => 'required|email|unique:users',

        'password' => 'required|min:6',

    ]);

    if ($validator->fails()) {

        return response(['error' => $validator->errors()], 401);

    }

    $input = $request->all();

    $input['password'] = bcrypt($input['password']);

    $user = User::create($input);

    $token = $user->createToken('MyAppToken')->accessToken;

    return response(['user' => $user, 'access\_token' => $token]);

}

Шаг 8: Обновление middleware

В вашем файле app/Http/Kernel.php добавьте 'auth:api' middleware к api группе middleware:

        'api' => [

            'throttle:api',

            'auth:api',

        ],

Это защитит ваши API-маршруты и потребует аутентификации для доступа.

Шаг 9: Запуск миграций

Запустите миграции Passport для создания необходимых таблиц:

php artisan migrate

Шаг 10: Готово!

Теперь ваше API использует аутентификацию с помощью Laravel Passport и токенов. Вы можете создавать, регистрировать и аутентифицировать пользователей с помощью API-запросов.

Не забудьте создать маршруты и методы контроллера для входа, выхода и регистрации пользователей в соответствии с вашими потребностями.

Материал к уроку:

1. <https://www.passportjs.org/>
2. <https://habr.com/ru/articles/441946/>
3. <https://www.toptal.com/laravel/restful-laravel-api-tutorial>

Домашнее задание:

Задание: Создание Full Stack веб-приложения на базе Laravel с AJAX

Цель этого домашнего задания - разработать полноценное веб-приложение, которое включает в себя серверную и клиентскую части, с использованием Laravel для API и AJAX-запросов на клиентской стороне.

Часть 1: Backend (Laravel API)

* Создайте новый проект Laravel и настройте его.
* Создайте модели, миграции и контроллеры для двух связанных сущностей (например, "Посты" и "Комментарии").
* Разработайте API с использованием Laravel для выполнения следующих операций:
* Создание, чтение, обновление и удаление постов.
* Создание, чтение, обновление и удаление комментариев к постам.
* Реализуйте аутентификацию и авторизацию пользователей для защиты API-маршрутов.
* Документируйте API с использованием инструментации, такой как Swagger или Postman.

Часть 2: Frontend (AJAX и JavaScript)

* Создайте папку "frontend" и разработайте клиентскую часть вашего приложения с использованием HTML, CSS и JavaScript.
* Создайте страницы для просмотра списка постов, деталей поста и добавления комментариев.
* Используйте AJAX-запросы для связи с вашим API Laravel и выполнения следующих действий:
* Загрузка списка постов.
* Создание нового поста.
* Загрузка деталей поста, включая список комментариев.
* Добавление нового комментария к посту.
* Стилизуйте ваше приложение с использованием CSS и, по желанию, фреймворка для фронтенда (например, Bootstrap).
* Реализуйте пользовательский интерфейс для аутентификации (входа и выхода) с использованием AJAX-запросов.

Задание для сдачи:

* Загрузите ваш проект Laravel на GitHub, включая код для API и клиентской части.
* В README.md предоставьте инструкции по установке и запуску вашего приложения.
* Убедитесь, что ваше приложение полностью функционально и обеспечивает CRUD-операции для постов и комментариев, а также аутентификацию пользователя на клиентской и серверной стороне.

