Фреймворк Laravel

Создание простой игры «камень-ножницы-бумага».

План работы:

- создать новый проект;
- организовать базу данных, чтобы сохранять статистику игр;
- сделать миграцию базы данных;
- добавить маршрут;
- создать контроллер;
- создать кнопки для игры;
- оформить всё с помощью CSS-стилей.

Создать проект Laravel

Установить Laravel через Composer, который автоматически умеет скачивать Laravel и все зависимости.

Откройте командную строку и введите следующую команду:

```
composer create-project --prefer-dist laravel/laravel rock-paper-scissors
```

Откройте папку с проектом в Visual code. Создайте терминал.

Создать базу данных

Создайте в phpMyAdmin базу данных для этого проекта и подключите ее в файле .env (строки 11-16).

Создать миграцию

В терминале пишем выполняем команду:

```
php artisan make:migration create_game_statistics_table
```

Открыть файл миграции (он находится в папке в папке `database/migrations`) и задать структуру таблицы:

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateGameStatisticsTable extends Migration
{
    public function up()
    {
        Schema::create('game_statistics', function (Blueprint $table) {
            $table->id();
            $table->string('player_name');
            $table->string('computer_choice');
            $table->string('player_choice');
            $table->string('result');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('game_statistics');
    }
}
```

Запустить миграцию командой

`php artisan migrate`

Добавить маршрут

Нам понадобятся два маршрута: корневой / и игровой /play.

Маршруты будем добавлять в файл web.php.

```
Route::get('/', [GameController::class,'index'])->name('game.index');
Route::post('/play', [GameController::class,'play'])->name('game.play');
```

На корневом маршруте мы будем отображать статистику и предлагать пользователю сыграть в игру, а на игровом маршруте будем собственно играть.

Этот код будет работать так: если мы заходим по корневому маршруту /, то PHP отрисует страницу `game.index`, а если зайдём по игровому маршруту `/play`, то он создаст страницу `game.play`.

В начало php-сценария необходимо добавить ссылку на папку с контроллерами:

```
namespace App\Http\Controllers;
```

Создать контроллер

```
php artisan make:controller GameController
```

Открыть файл с контроллером (он находится в папке `app/Http/Controllers`), внести в него следующий код:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class GameController extends Controller
{
    public function index()
    {
        return view('game');
    }
}
```

Этот код определяет метод `index`(), который будет возвращать игровое представление, чтобы отображать элементы на сайте. Мы опишем его на следующем шаге.

А пока добавим в этот же файл ещё одну функцию — `play`():

```
public function play(Request $request)
{
    $playerChoice = $request->input('choice');

    // Генерируем случайный ход для компьютера
    $computerChoice = rand(1, 3);
    if ($computerChoice == 1) {
        $computerChoice = 'rock';
    } elseif ($computerChoice == 2) {
        $computerChoice = 'paper';
    } else {
        $computerChoice = 'scissors';
    }

// Получаем результат игры
    if ($playerChoice == $computerChoice) {
        $result = 'tie';
    } elseif (($playerChoice == 'rock' && $computerChoice == 'scissors')
|| ($playerChoice == 'paper' && $computerChoice == 'rock') ||
($playerChoice == 'scissors' && $computerChoice == 'paper')) {
        $result = 'win';
    } else {
        $result = 'lose';
    }

    // Сохраняем статистику в базу данных
    DB::table('game_statistics')->insert([
        'player_name' => 'Player 1',
        'computer_choice' => $computerChoice,
        'player_choice' => $playerChoice,
        'result' => $result,
        'created_at' => now(),
        'updated_at' => now(),
    ]);

    // Возвращаем результат игры в представление
    return view('game', ['result' => $result]);
}
```

Здесь мы прописали основную логику игры. На вход функция принимает объект Request, который содержит ход игрока. Экземпляр этого объекта будет приходить, когда мы нажмём на одну из кнопок в игре («Камень», «Ножницы» или «Бумага»), а потом отправим наш выбор методом POST.

После того как мы обработали запрос, нам нужно сгенерировать ответ компьютера — с помощью генератора случайных чисел. И в конце мы определяем победителя и записываем результат в базу данных методом `insert()`.

Логика игры готова.

Создать кнопки для игры

Метод index() из файла GameController.php отображает страницу с игрой, но пока на ней ничего нет, поэтому давайте создадим пару кнопок.

Открываем папку resources/views и создаём там файл с названием game.blade.php. Добавляем туда следующий код, чтобы создать игровой интерфейс:

```
<!DOCTYPE html>
<html>
<head>
    <title>Rock Paper Scissors</title>
</head>
<body>
    <h1>Rock Paper Scissors</h1>

    @if(isset($result))
        <p>You {{ $result }}!</p>
    @endif

    <form method="POST" action="{{ route('game.play') }}">
        @csrf
        <label for="rock">Rock</label>
        <input type="radio" name="choice" id="rock" value="rock">
        <label for="paper">Paper</label>
        <input type="radio" name="choice" id="paper" value="paper">
        <label for="scissors">Scissors</label>
        <input type="radio" name="choice" id="scissors" value="scissors">
        <button type="submit">Play</button>
    </form>
</body>
</html>
```

Мы создали простой игровой интерфейс, который состоит из трёх кнопок с выбором вариантов. Также мы добавили кнопку, чтобы отправлять данные в GameController и получать результат игры $result, если она уже была сыграна.

Результат игры нужен, чтобы мы могли видеть, кто победил в прошлой игре. А чтобы постоянно не выводить этот текст, мы сперва проверяем, была ли переменная $result объявлена через GameController.

Теперь мы можем запустить наш сервер и посмотреть на результат.

Если нажать на одну из кнопок, можно узнать, кто победил.

Но сразу проявляется одна проблема: мы не знаем, какие ходы сделал компьютер, а какие — мы. Давайте это исправим.

Заходим обратно в файл GameController.php и изменяем всего одну строку:

```
return view('game', ['result' => $result, 'player_choice' =>
$playerChoice, 'computer_choice' => $computerChoice]);
```

Теперь мы будем возвращать в наше представление дополнительные переменные. Внесём изменения и в файл game.blade.php:

```
@if(isset($result))
    <p>You played {{ $player_choice }}</p>
    <p>Computer played {{ $computer_choice }}</p>
    <p>You {{ $result }}!</p>
@endif
```

Перезапускаем сервер и видим, что теперь всё работает корректно.

Осталось вывести результаты последних игр, и наше приложение будет готово. Для простоты будем учитывать статистику последних десяти игр.

Для этого необходимо изменить файл GameController.php, а вернее функции index() и play():

```
public function index()
    {
        $last_games = DB::table('game_statistics')
                ->orderBy('created_at', 'desc')
                ->limit(10)
                ->get();

        return view('game', ['last_games' => $last_games]);
    }

    public function play(Request $request)
    {
        $last_games = DB::table('game_statistics')
                ->orderBy('created_at', 'desc')
                ->limit(10)
                ->get();
        $playerChoice = $request->input('choice');

        ...

        // Возвращаем результат игры в представление
        return view('game', ['result' => $result, 'player_choice' =>
$playerChoice, 'computer_choice' => $computerChoice, 'last_games' =>
$last_games]);
    }
```

Мы опустили несколько строк, чтобы показать лишь самые важные фрагменты. Как видите, у нас появилась новая переменная $last_games, в которую помещаются 10 последних строк базы данных. Делается это с помощью двух методов: orderBy(), который используется, чтобы доставать только самые свежие игры, и limit(10), который помогает ограничить их количество десятью.

В каждом представлении мы также добавляем этот список в качестве возвращаемого значения.

Теперь нужно поправить файл game.blade.php и отобразить эти игры:

```
<ul>
    @foreach ($last_games as $last_game)
        <li>
            <p>You {{ $last_game->result }}</p>
            <p>You played {{ $last_game->player_choice }}</p>
            <p>Computer played {{ $last_game->computer_choice }}</p>
        </li>
    @endforeach
</ul>
```

Этот код просто создаёт ненумерованный список, который располагается после формы. Внутри списка мы добавляем элементы <li> через цикл @foreach. В нём мы проходим по всем нашим играм и выводим для каждой игры результат, ход игрока и компьютера.

Сохраним файлы и обновим приложение: на странице выводятся результаты завершённых игр, а новые игры успешно сохраняются.

Оформление с помощью CSS-стилей

Откроем папку public и создадим файл styles.css. Добавим в него следующий код:

```css
body {
    font-family: Arial, sans-serif;
    background-color: #f5f5f5;
}

h1 {
    text-align: center;
    margin-top: 50px;
}
```

```css
form {
    display: flex;
    flex-direction: column;
    align-items: center;
    margin-top: 30px;
}

label {
    margin-right: 70px;
    margin-top: 20px;
    font-size: 20px;
}
```

```css
input[type="radio"] {
    margin-left: 10px;
    padding-bottom: 50px;
}

button[type="submit"] {
    margin-top: 20px;
    background-color: #008CBA;
    color: white;
    border: none;
    padding: 10px 20px;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

button[type="submit"]:hover {
    background-color: #006B9F;
}
```

```css
p {
    text-align: center;
    margin-top: 20px;
    font-size: 24px;
    font-weight: bold;
}

ul {
    list-style-type: none;
}

li {
    background: #dadada;
}
```

Откроем файл game.blade.php, чтобы подключить в него новые стили:

```
<!DOCTYPE html>
<html>
<head>
    <title>Rock Paper Scissors</title>
    <link rel="stylesheet" href="{{ asset('styles.css') }}">
</head>
<body>
    ...
</body>
</html>
```

Приложение готово.