Laravel лекція №2.

Тема: Створення додатку за допомогою фреймворку Laravel

План:

1. Зручність Laravel.
2. Встановлення.
3. Розробка гри «Камінь, ножиці, папір».

Зручність Laravel.

Laravel «з коробки» містить:

- Інтерактивна документація Для кожної функції є окрема стаття із прикладами. Дуже зручно як для початківців, так і для досвідчених розробників.
- Міграція бази даних. Laravel надає зручний спосіб працювати з базами даних за допомогою міграцій. Міграції дають змогу змінювати структуру бази даних без необхідності писати SQL-запити.
- Artisan. Консоль для розробників, яка полегшує взаємодію з Laravel, дозволяє проводити міграції баз даних, налаштовувати авторизацію та взаємодіяти з компонентами фреймворку.
- ORM (object-relational mapping). Eloquent ORM дозволяє працювати з базами даних в об'єктно-орієнтованому стилі, тобто вибудовувати взаємодію з повноцінними об'єктами, якими можна маніпулювати.
- Шаблони. У Laravel купа шаблонів для створення інтерфейсу користувача. Це значно полегшує процес розробки.
- Реєстрація та авторизація. Фреймворк надає готові шаблони для аутентифікації користувачів, щоб розробникам не доводилося винаходити колесо.
- Зручний дебаггінг та тестування додатків. Зазвичай у веб-розробці не дуже зручно реалізовано перевірку коду на надійність, тому творці Laravel вирішили спростити цей процес.

На цьому перелік можливостей Laravel не закінчується. Ще є більш специфічні штуки: кешування, маршрутизація, MVC та багато інших корисних фіч. Але головне, що варто знати про Laravel, - на ньому можна швидко та зручно створювати сайти.

Переваги Laravel інших PHP-фреймворків

Якщо порівнювати Laravel з іншими популярними фреймворками, то такі відмінності:

- Найменша складність. Laravel простіше вивчити і використовувати, ніж, наприклад, Symfony. Але така простота не обмежує його функціональність.
- Зручна маршрутизація. Laravel пропонує зручну маршрутизацію, наприклад, дозволяючи групувати, кешувати та давати власні назви маршрутам, а також визначати дії на різні HTTP-запити.
- Інтеграція із бібліотеками. Laravel використовує пакетний менеджер Composer, який дозволяє швидко підключати сторонні бібліотеки до проекту без зайвих проблем.
- Сучасні архітектури. У Laravel вбудована сучасна архітектура, включаючи шаблон MVC (Model-View-Controller).

В цілому Laravel дотримується відмінного балансу між функціональністю, гнучкістю і простотою у використанні. Щоправда, і він має свої особливості та межі застосування. Тому його не можна назвати «найкращим PHP-фреймворком». Однак творці Laravel намагалися зробити все, щоб він був комфортним для розробки:

> *«Laravel призначений, щоб усунути болі в розробці шляхом спрощення звичних завдань, які часто зустрічаються в веб-проектах… Laravel потрібен, щоб зробити процес розробки приємним для програмістів без погіршення функціональності додатків. Як кажуть, щасливі розробники пишуть найкращий код».*
>
> ***АВТОРИ LARAVEL***

Якщо Laravel - це популярний фреймворк для розробки веб-додатків, то на ньому пишуть різні веб-додатки:

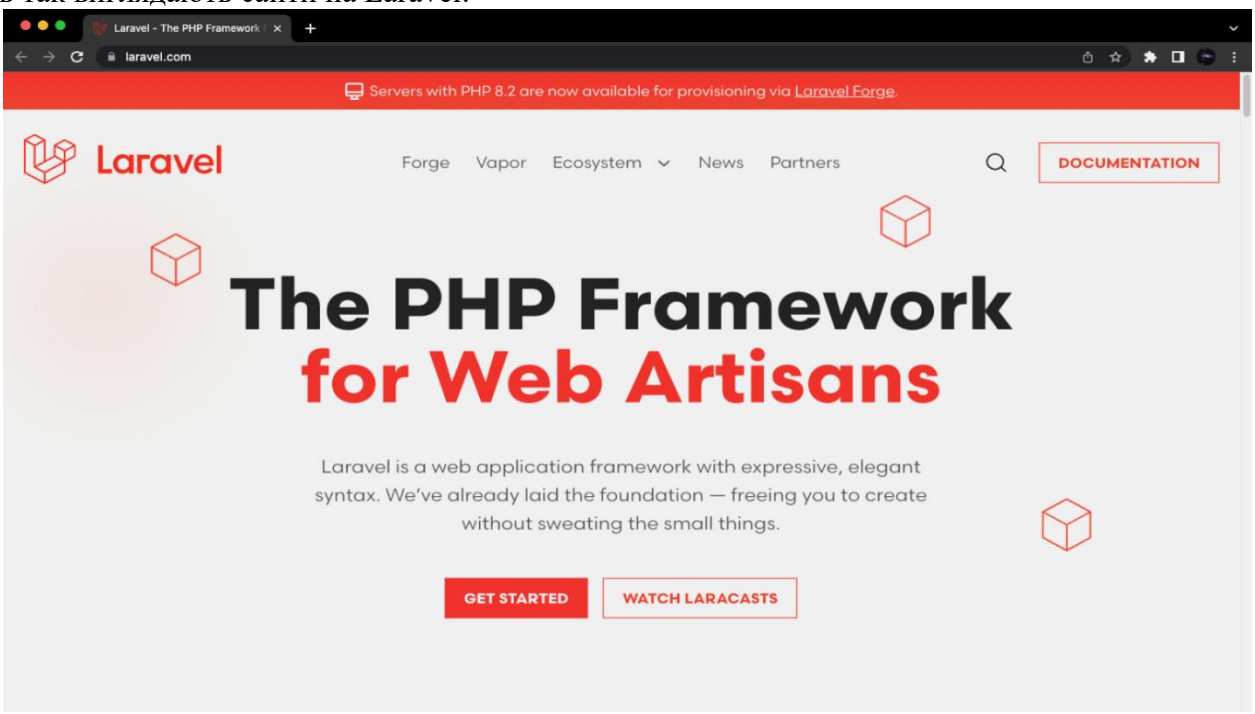
- Лендінги, або посадкові сторінки. На Laravel можна з мінімальними витратами написати як просту сторінку, так і багатосторінковий сайт для компанії. Звичайно, це буде складніше, ніж використовувати Tilda, але можна масштабувати сайт і додавати будь-які власні фічі. Приклад: лендінг Canva .
- API Це прошарок для обміну даними між клієнтською та серверною сторонами веб-програми .
- CMS. Системи керування контентом (content management system, або CMS) - вони потрібні, щоб керувати вмістом веб-сайту. Приклад: October CMS .

- Онлайн-магазини. На Laravel створюють електронні комерційні платформи, на яких можна продавати товари та послуги. Приклад: Bagisto .

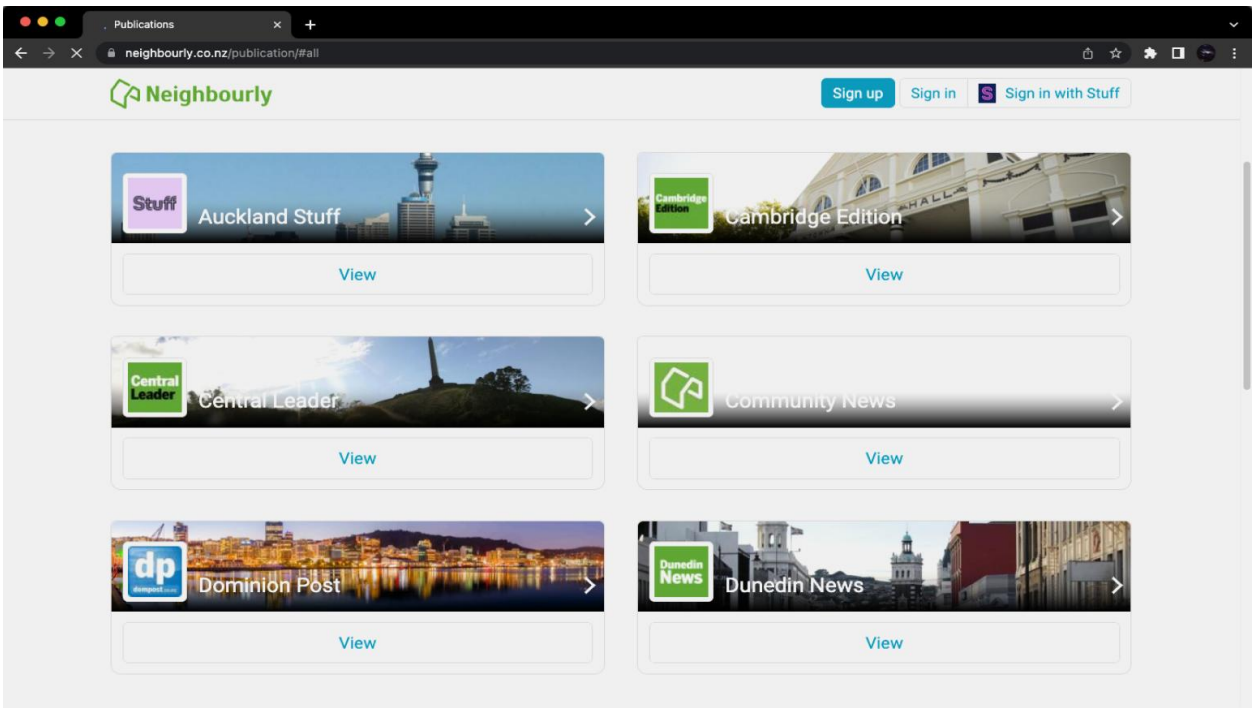
Ось ще кілька великих проектів, які написані на Laravel:

- laravel.com - офіційний сайт Laravel створений на Laravel;
- Neighbourly – соціальна мережа для спілкування з сусідами;
- World Walking - додаток для мотивації людей ходити більше;
- Ko-fi – платформа для підтримки творчості;
- The Invoice Machine – онлайн-сервіс для виставлення рахунків;
- Startups.com - платформа для запуску та розвитку стартапів;
- Larametrics – це інструмент для моніторингу та аналізу продуктивності додатків на Laravel.

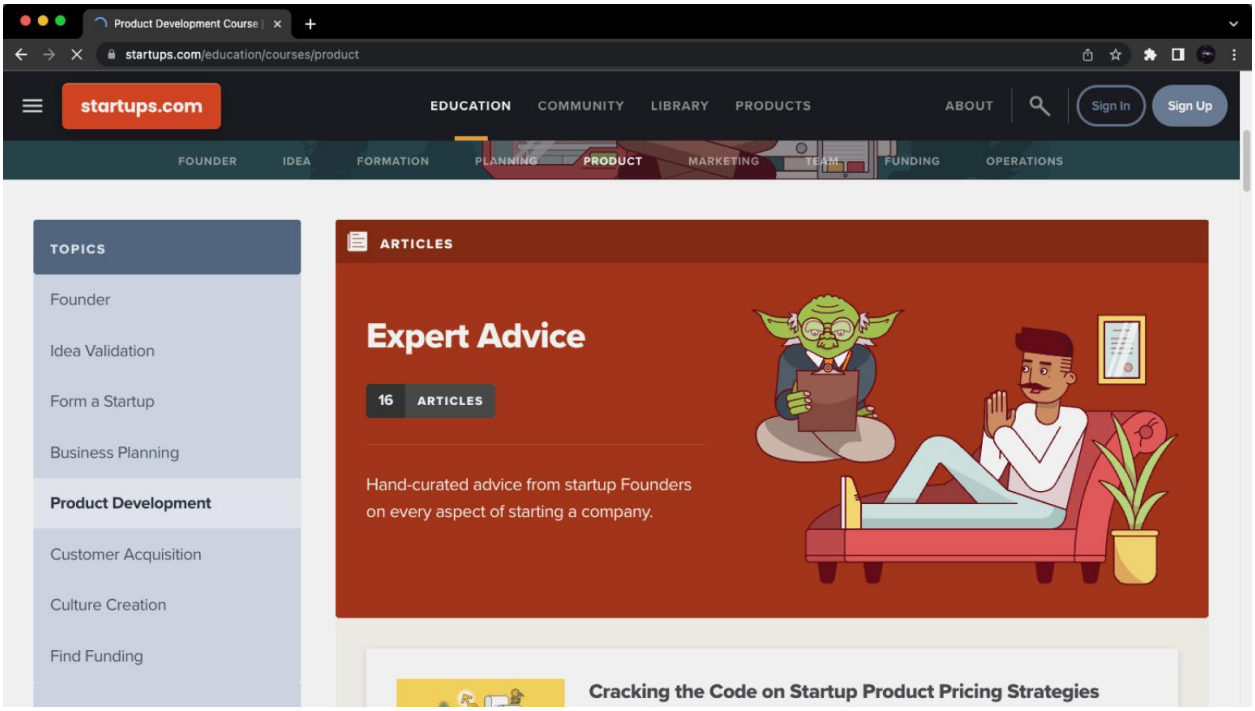
А ось так виглядають сайти на Laravel:



Офіційний сайт Laravel



Соціальна мережа Neighbourly



Платформа для запуску та розвитку стартапів — Startups.com

Це лише частина сайтів, написаних на Laravel. Деякі платформи можуть використовувати фреймворк як один із компонентів складного веб-програми, але зазвичай про це не говорять — особливо часто відмовчуються великі компанії.

З чого розпочати роботу

Для прикладу буде розглянуто створення грального додатку «Камінь, ножиці, папір».

2. Встановлення

Але для початку нам треба буде встановити Laravel та MySQL .

Завантажуємо Composer

Composer – це пакетний менеджер для PHP. За допомогою нього можна встановлювати сторонні бібліотеки та фреймворки, як це потрібно в нашому випадку. Розглянемо установку для операційних систем Windows, MacOS та Linux.

Windows. На цій ОС процес установки найпростіший:

- переходимо на офіційний сайт Composer і завантажуємо інсталяційний файл Composer-Setup.exe ;
- запускаємо його та проходимо звичні кроки установки.

macOS. Встановити Composer безпосередньо не вийде, спочатку потрібно завантажити менеджер пакетів Homebrew . Відкриваємо термінал та вводимо команду:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Після встановлення Homebrew вводимо команду:

```
brew install composer
```

Linux. Тут встановити Composer простіше, ніж macOS. Для різних Linux-систем команди можуть відрізнятися (через особливості системи управління пакетами), але процес установки, по суті, буде ідентичним. Ось як виглядає процес установки в Debian-подібних дистрибутивах:

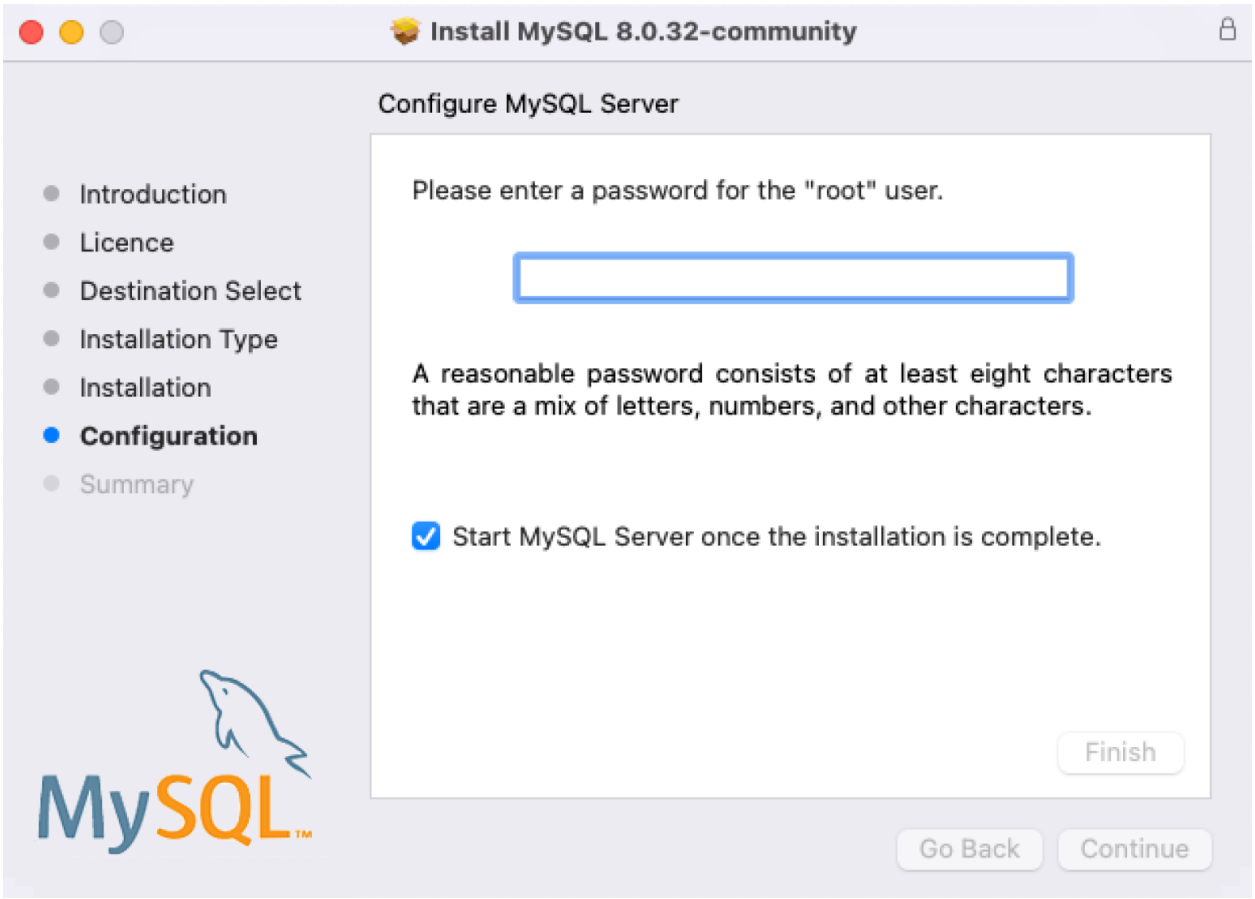
```
$ sudo apt-get update
$ sudo apt-get install composer
```

Якщо виникли труднощі із встановленням, можна звернутися до документації на офіційному сайті . А якщо все вийшло – переходимо далі.

Завантажуємо MySQL

Ми зберігатимемо статистику ігор базі даних, а нею потрібен власний сервер. Тому скачуємо MySQL з офіційного сайту .

Далі треба вибрати операційну систему і пройти стандартний процес установки.



Первинна настройка MySQL

Залиште галочку, де кажуть, що MySQL Server запуститься одразу після встановлення.

Альтернативне встановлення для Linux. Якщо ви використовуєте Linux, то завантажувати установник не обов'язково - можна знову скористатися пакетним менеджером. Робиться це приблизно так (з поправкою на ваш дистрибутив):

```
sudo apt-get install mysql-server
```

Після встановлення потрібно запустити сам сервер:

```
sudo service mysql status
```

Готово – сервер запущений!

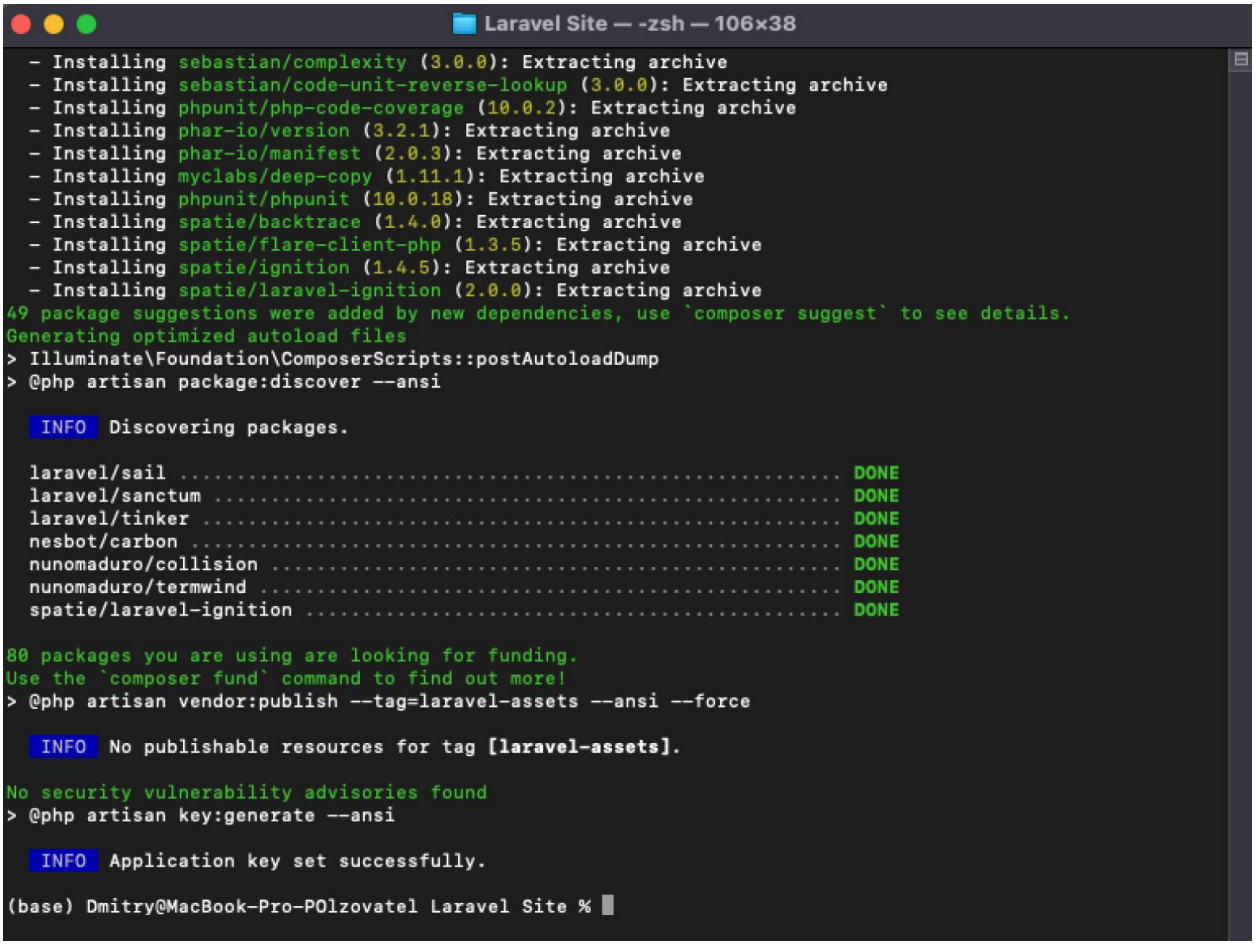
Встановлюємо Laravel

Тепер нам потрібно встановити Laravel. Це можна зробити через Composer, який автоматично вміє завантажувати Laravel та всі залежності.

Відкрийте командний рядок та введіть наступну команду:

```
composer create-project --prefer-dist laravel/laravel myapp
```

Тут myapp це ім'я вашого проекту. Ця команда створить новий проект Laravel у папці myapp .



Створення нового Laravel-проекту

Ми встановили Laravel останньої версії. На скріншоті показаний успішний процес установки. Якщо виникли проблеми — дивіться документацію щодо встановлення на офіційному сайті .

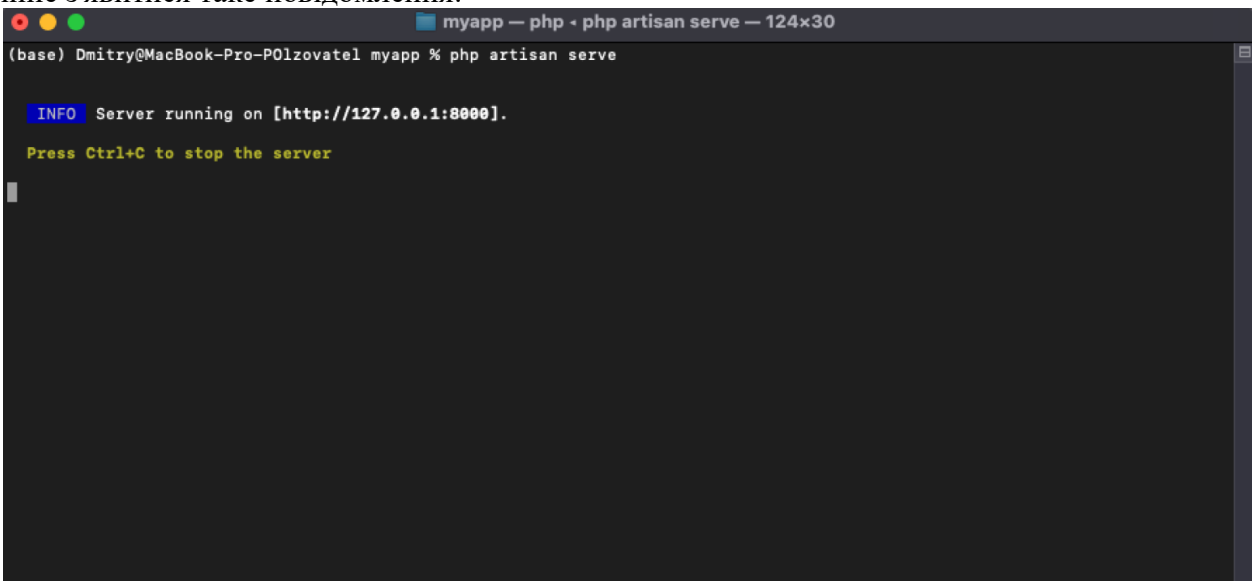
Запускаємо сервер

Щоб перевірити правильність установки Laravel, нам достатньо лише запустити вбудований веб-сервер. До речі, не рекомендуємо використовувати вбудований веб-сервер у продакшені – для реального проекту краще вибрати Apache або Nginx .

Запустити веб-сервер просто: переходимо до папки з проектом і відкриваємо командний рядок.

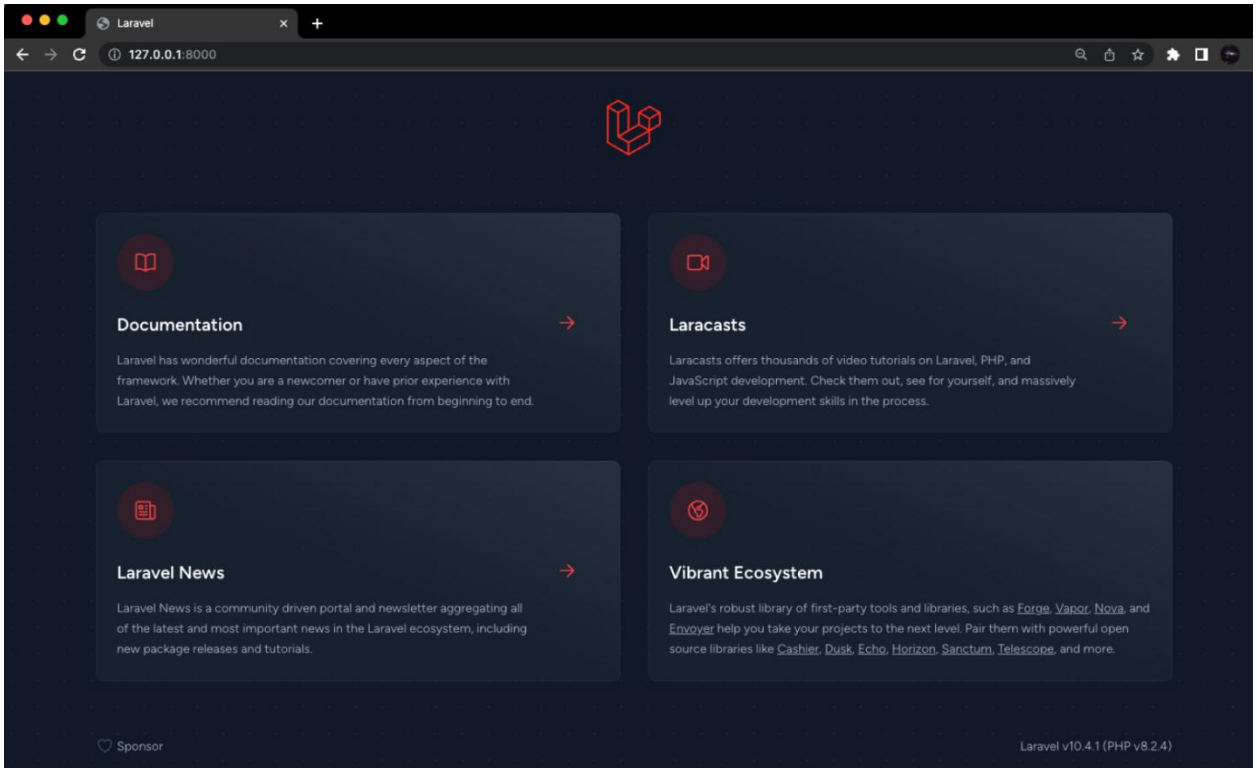
```
php artisan serve
```

Повинне з'явитися таке повідомлення:



Веб-сервер успішно запущено

Відкриваємо браузер та переходимо за вказаною адресою: http://127.0.0.1:8000/ . Якщо все добре, має відкритися приблизно такий екран:



Сторінка за замовчуванням

Це сторінка Laravel за промовчанням. Її, звичайно ж, можна міняти — цим ми й займатимемося далі. Ну а поки що буде корисно перейти за посиланнями — почитати документацію та свіжі новини про фреймворк.

Як створити веб-додаток на Laravel

Тепер, коли у нас встановлено Laravel, можна розпочинати написання коду. Як ми вже згадували вище, ми створимо гру «Камінь, ножиці, папір» і гратимемо в неї з комп'ютером.

План такий:

- Створити новий проект та вибрати назву.
- Організувати базу даних для збереження статистики ігор.
- Зробити міграцію бази даних щоб всі таблиці були на своєму місці.
- Додати маршрут та розібратися, що це таке.
- Створити контролер і також зрозуміти, навіщо він потрібний.
- Створити кнопки для гри .
- Прикрасити все за допомогою CSS-стилів .

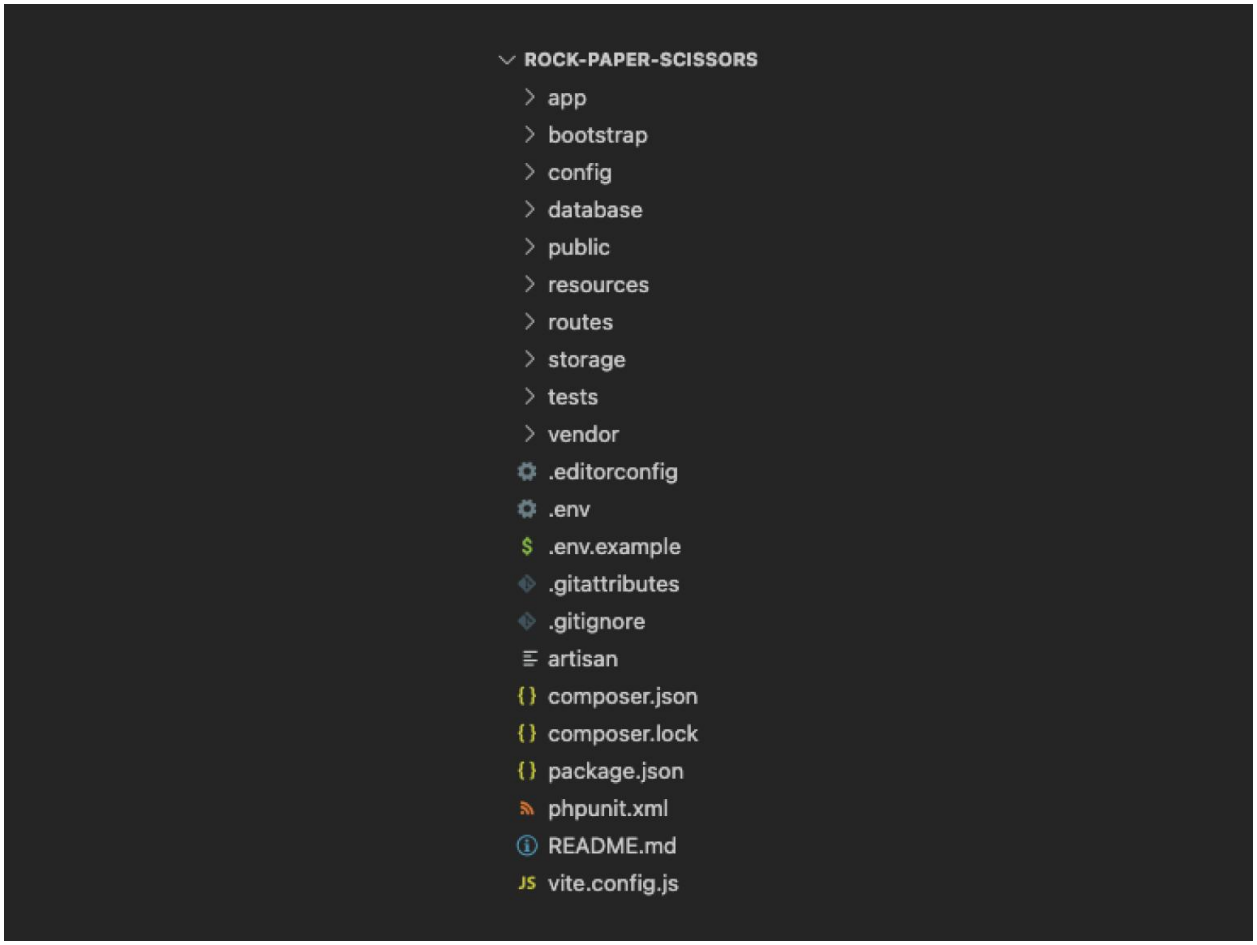
План здоровий - приступимо!

Створюємо новий проект

Ідемо вже знайомим шляхом: запускаємо командний рядок та вводимо команду для Composer:

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

Проект створили - файли не змінилися:



Структура проекту

Залиште командний рядок відкритим — він нам ще стане в нагоді, а самі перейдіть в папку з проектом за допомогою наступної команди:

```
cd rock-paper-scissors
```

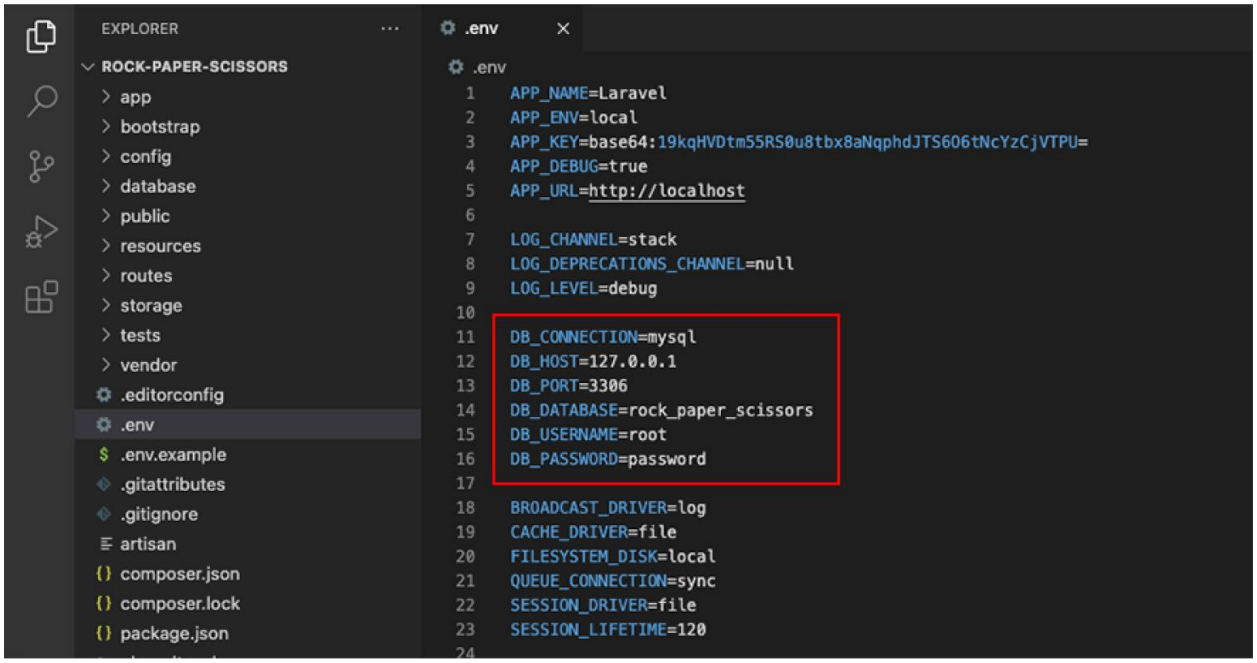
Команда cd дозволяє рухатися вгору та вниз папками. Ну, а раз ми вже в потрібному місці, переходимо далі.

Організуємо базу даних

Щоб зберігати результати ігор, нам потрібна база даних. Вона є звичайним файлом, у якому дані зберігаються у зручному для використання вигляді.

Перейдемо у файл .env , який знаходиться у папці нашого проекту. Він може бути прихований, тому що його назва починається з точки. Якщо він не відображається, змініть параметри відображення файлів у своїй системі.

У файлі нам потрібно виправити такі рядки:



Нам потрібні останні три опції:

```
DB_DATABASE=rock_paper_scissors
DB_USERNAME=root
DB_PASSWORD=password
```

У першому рядку ми задаємо назву бази даних, а в другому і третьому - дані для входу до неї.

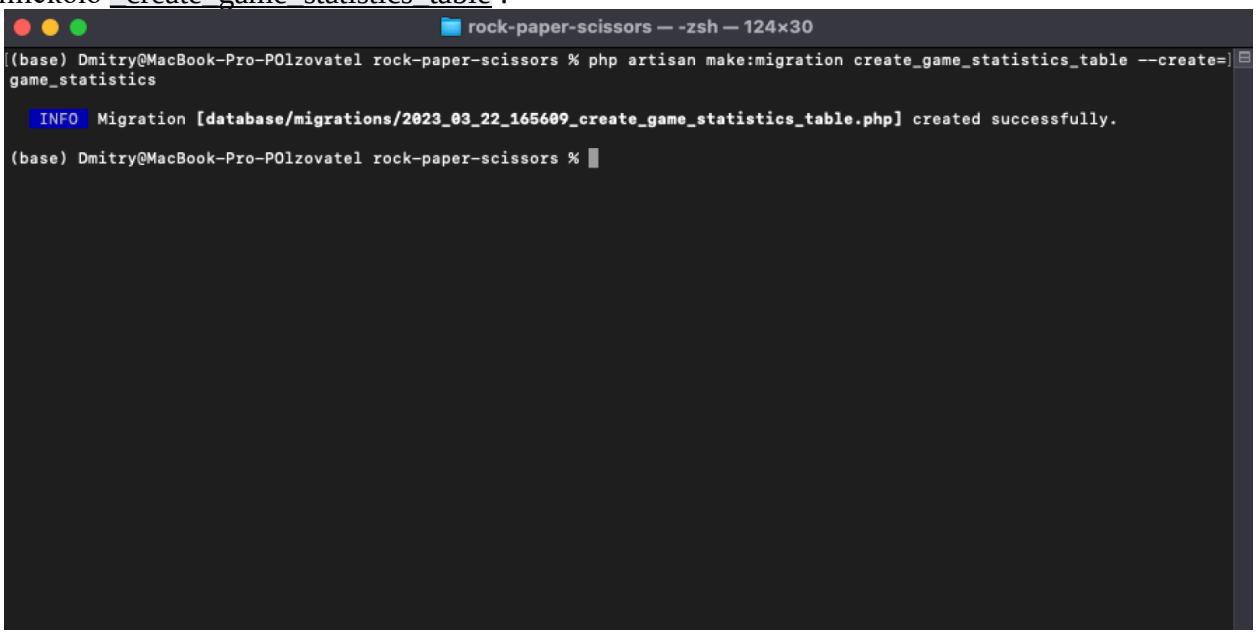
Зберігаємо файл і закриваємо його – база даних попередньо налаштована.

Робимо міграцію

Ми створили базу даних, настав час створити необхідні таблиці. Відкриваємо командний рядок і переконуємося, що у папці з проектом. Вводимо наступну команду:

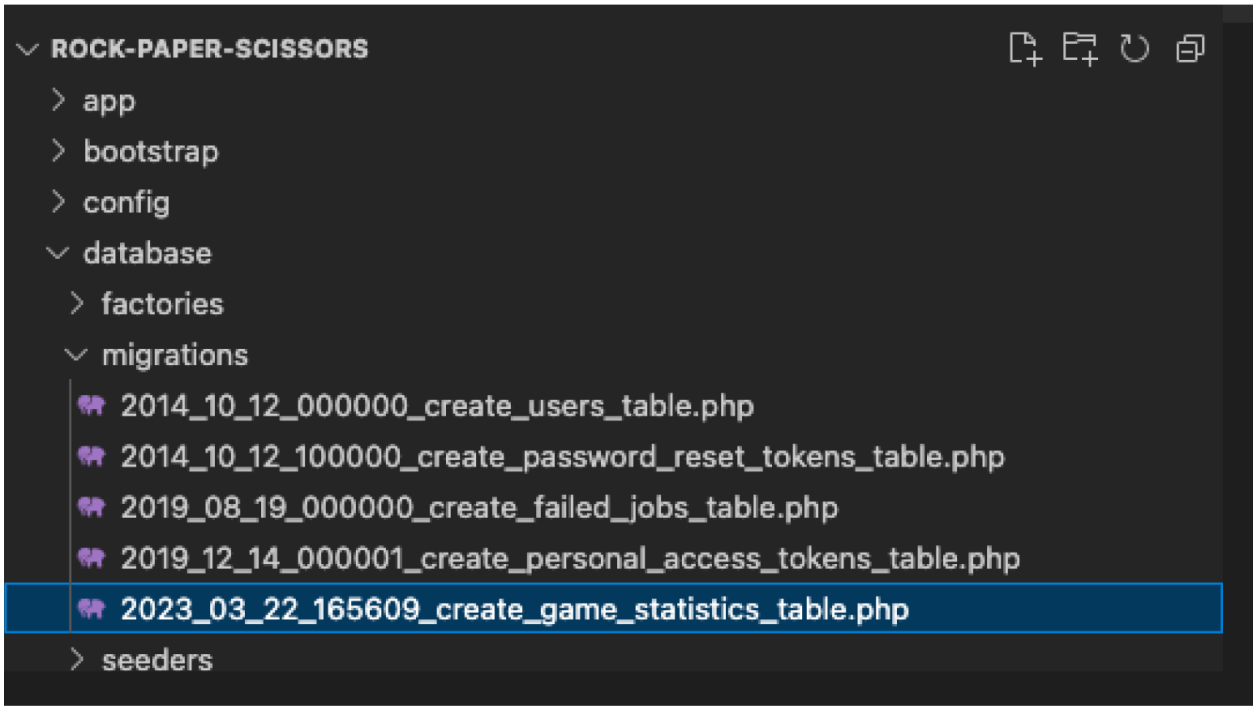
```
php artisan make:migration create_game_statistics_table --create=game_statistics
```

Ця команда зробить міграцію бази даних, тобто змінить її вміст та створить файл із таблицею. У нашому випадку - додасть таблицю game_statistics , а також файл з датою створення таблиці та припискою _create_game_statistics_table :



Створюємо нову таблицю в БД

З'явився PHP-файл міграції, який знаходиться у папці database/migrations :



Міграції у структурі Laravel-проекту

Відкриваємо його і додаємо наступний код – він допоможе задати структуру нашої таблиці:

```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');
    }
}
```

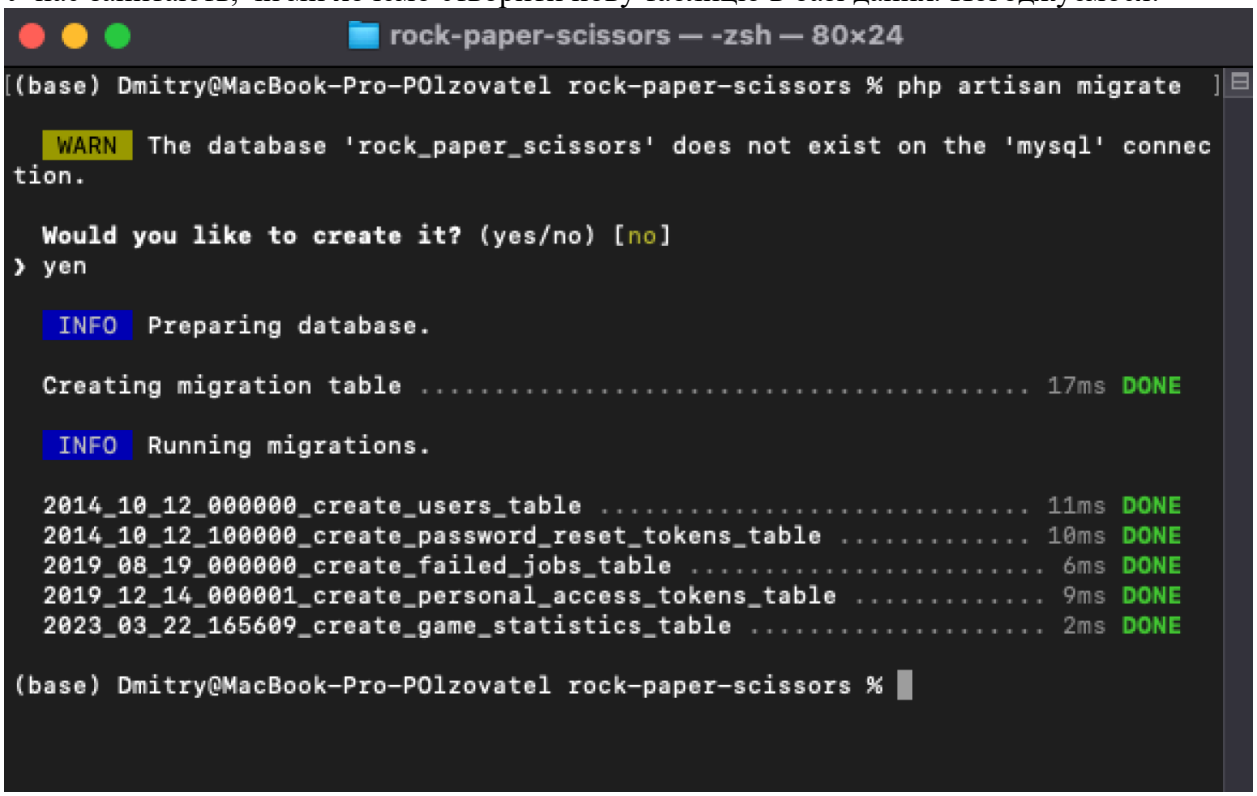
Ця міграція створить таблиці з шістьма колонками: унікальним ідентифікатором гри (id), ім'ям гравця (player_name), ходом комп'ютера (computer_choice), ходом гравця (player_choice), результатом гри (result) і тимчасовою відміткою (timestamps) - вказано (timestamps) -

Зверніть увагу на типи даних у нашій таблиці. У нас є чотири колонки з типом рядок (player_name , computer_choice , player_choice і result), а також два унікальні типи даних: id і timestamps . Перший — це айдішник рядка в таблиці, і він є звичайним числом, але з додатковими фічами. А другий є датою у форматі YYYY-MM-DD HH:MM:SS .

Зберігаємо файл та закриваємо його. Тепер нам потрібно запустити міграцію – термінал нам допоможе:

```
php artisan migrate
```

У нас запитають, чи ми хочемо створити нову таблицю в базі даних. Погоджуємося:



База даних настроєна та готова до роботи

Готово — тепер ми маємо працюючу базу даних з потрібною таблицею.

Додаємо маршрут

Весь бекенд будується на маршрутах або URL-адресах. Вони допомагають задавати зручну структуру та зрозумілу поведінку веб-додатків.

Для користувача маршрути – це окремі вкладки на сайті. Наприклад, якщо зайти на сайт Skillbox, відкриється головна сторінка ztu.edu.ua . А якщо натиснути на будь-який курс, ми перейдемо на іншу сторінку сайту з іншою URL-адресою, наприклад ztu.edu.ua/course/profession-python/ . Ми побачимо, що на адресу нашого сайту додався текст: /course/profession-python/ . Ця «приписка» і перенесла нас на іншу сторінку з іншим вмістом. Але не просто перенесла, а запустила певний код, який і змалював цю сторінку.

Нам знадобляться два маршрути: кореневий / та ігровий / play . Щоб їх додати, потрібно перейти до папки routes , відкрити файл web.php і вписати туди наступний код:

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

На кореневому маршруті ми відображатимемо статистику і пропонуватимемо користувачеві зіграти в гру, а на ігровому маршруті будемо власне грати.

Цей код буде працювати так: якщо ми заходимо кореневим маршрутом / , то PHP відобразить сторінку game.index , а якщо зайдемо по ігровому маршруту / play , то він створить сторінку game.play .

На початку потрібно прописати ще один рядок:

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

Підсумковий файл має виглядати так:

routes > web.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider and all of them will
| be assigned to the "web" middleware group. Make something great!
|
*/

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

Вміст файлу web.php

Готово! Не забуваймо зберегти файл.

Створюємо контролер

Контролер — це поняття шаблону проектування MVC.

Про контролерів простими словами

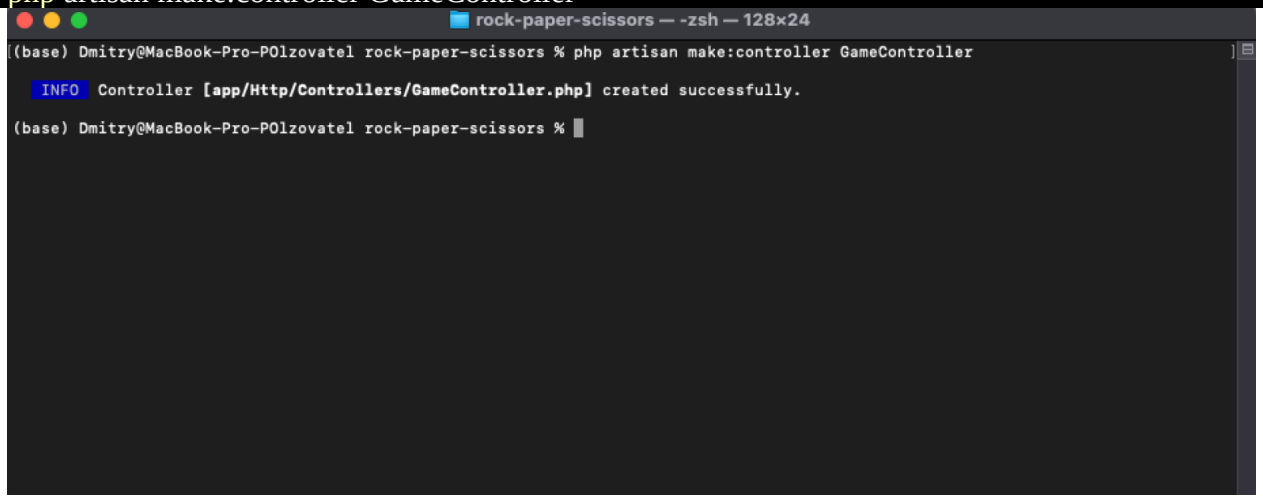
Щоб не вдаватися до подробиць, скажімо, що контролер — це частина програми, яка відповідає за обробку запитів від користувача та взаємодію з моделлю (даними) та поданням (відображенням).

Простими словами, контролер - це сполучна ланка між інтерфейсом користувача і базою даних. Коли користувач надсилає запит на сервер, контролер отримує цей запит, обробляє його та взаємодіє з базою даних, щоб отримати необхідні дані. Потім контролер передає ці дані у виставу, яка відображає їх на екрані для користувача.

Контролер відокремлює бізнес-логіку програми від інтерфейсу користувача, що дозволяє більш ефективно управляти додатком і забезпечує більш гнучку архітектуру. Він також дає можливість програмістам працювати з додатком окремо, без необхідності знати про інші компоненти програми.

Нам потрібно створити контролер для всієї гри. Називатися він буде GameController . Щоб створити його, потрібно запустити в терміналі наступну команду:

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



Ця команда створить новий файл із контролером у папці app/Http/Controllers . Перейдемо туди, відкриємо файл GameController.php і впишемо наступний 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');
    }
}
```

Цей код визначає метод index() , який повертатиме ігрову виставу, щоб відображати елементи на сайті. Ми опишемо його на наступному кроці.

А поки додамо в цей файл ще одну функцію — 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]);
}
```

Тут ми прописали головну логіку гри. На вхід функція приймає об'єкт Request , який містить перебіг гравця. Примірник цього об'єкта буде приходити, коли ми натиснемо на одну з кнопок у грі ("Камінь", "Ножиці" або "Папір"), а потім відправимо наш вибір методом POST .

Після того як ми обробили запит, нам потрібно згенерувати відповідь комп'ютера за допомогою генератора випадкових чисел. І наприкінці ми визначаємо переможця та записуємо результат до бази даних методом insert() .

Логіка гри готова! Знову ж таки, не забуваємо зберегти файл.

Створюємо кнопки для гри

Метод index() з файлу GameController.php відображає сторінку з грою, але поки що на ній нічого немає, тому давайте створимо пару кнопок.

Відкриваємо папку resources/views і створюємо там файл під назвою game.blade.php . Додаємо туди наступний код, щоб створити ігровий інтерфейс:

```
<!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>
```

Не дивуйтеся, що ми вставили HTML-код у PHP-файл, це нормально. PHP вміє працювати з HTML.

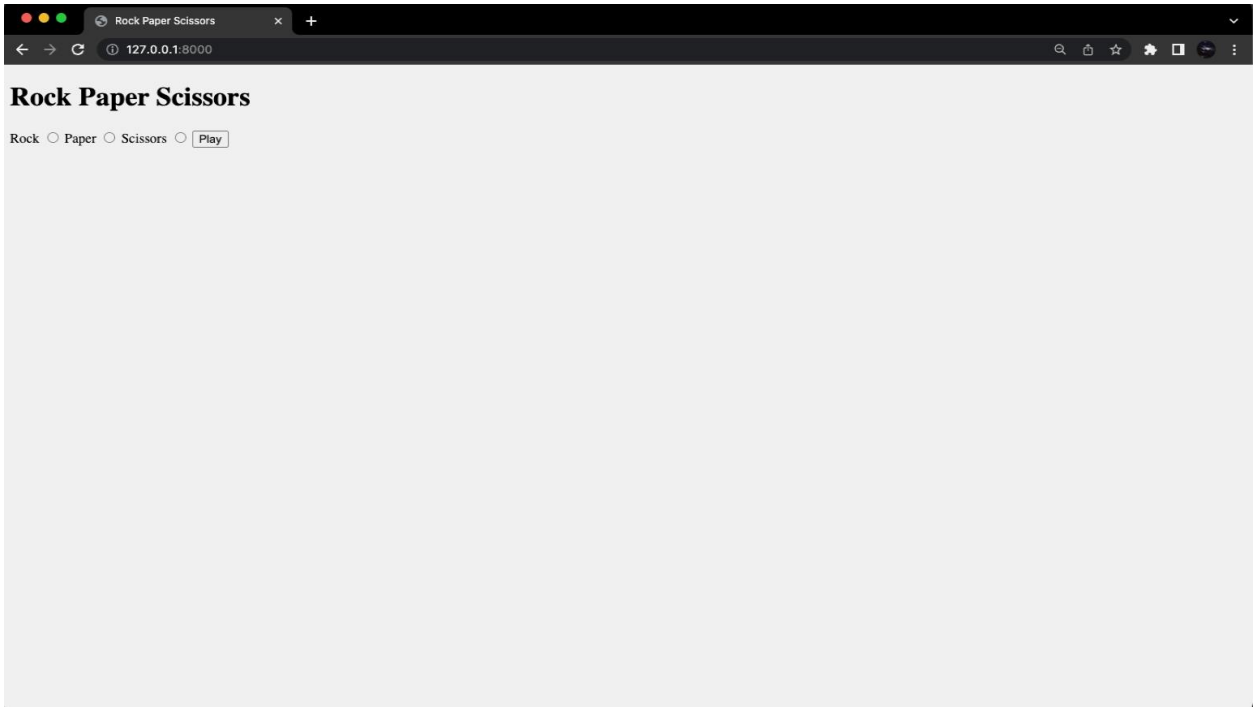
Але повернемося до коду: ми створили простий ігровий інтерфейс, який складається із трьох кнопок із вибором варіантів. Також ми додали кнопку, щоб відправляти дані в GameController і отримувати результат гри $result , якщо вона вже була зіграна.

Результат гри потрібен, щоб ми могли бачити, хто переміг у минулій грі. Щоб постійно не виводити цей текст, ми спочатку перевіряємо, чи була змінна $result оголошена через GameController .

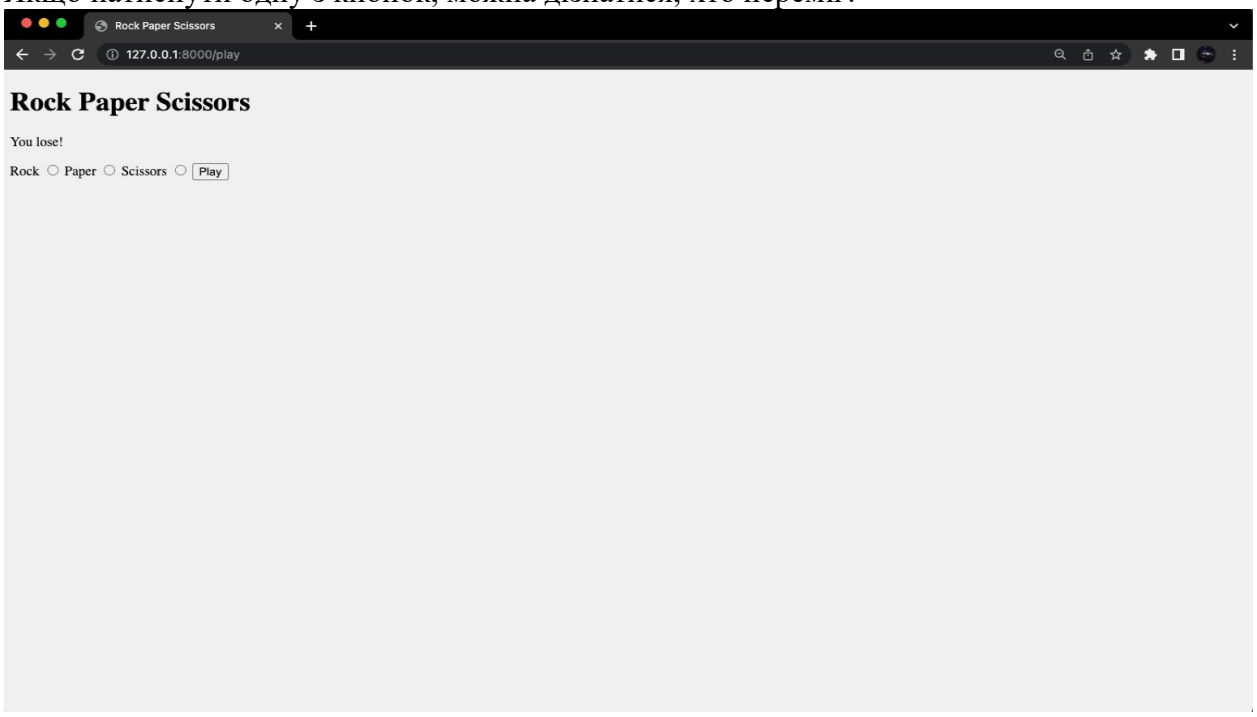
Тепер ми можемо запустити наш сервер і подивитися на результат:

```
php artisan serve
```

Бачимо те, що й хотіли побачити:



Якщо натиснути одну з кнопок, можна дізнатися, хто переміг:



Але одразу проявляється одна проблема: ми не знаємо, які ходи зробив комп'ютер, а які ми. Давайте це виправимо.

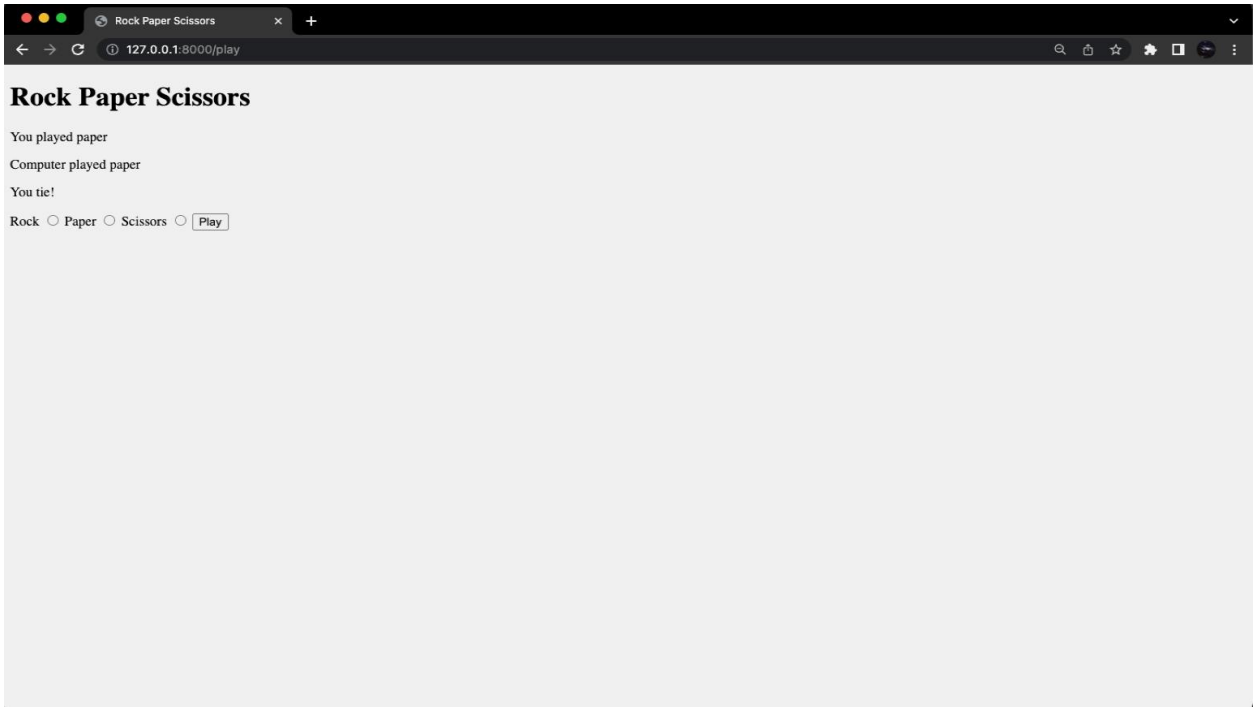
Заходимо назад у файл GameController.php і змінюємо лише один рядок:

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

Тепер ми будемо повертати до нашої вистави додаткові змінні. Внесемо зміни і до файлу game.blade.php :

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

Перезапускаємо сервер і бачимо, що тепер все працює коректно:



Залишилося вивести результати останніх ігор, і наша програма буде готова. Для простоти враховуватимемо статистику останніх десяти ігор.

Для цього необхідно змінити файл GameController.php , а вірніше функції index() та 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]);
    }
```

Ми опустили кілька рядків, щоб показати лише найважливіші фрагменти. Як бачите, у нас з'явилася нова змінна $ last_games , в яку поміщаються 10 останніх рядків бази даних .

У кожному поданні ми також додаємо цей список як значення, що повертається.

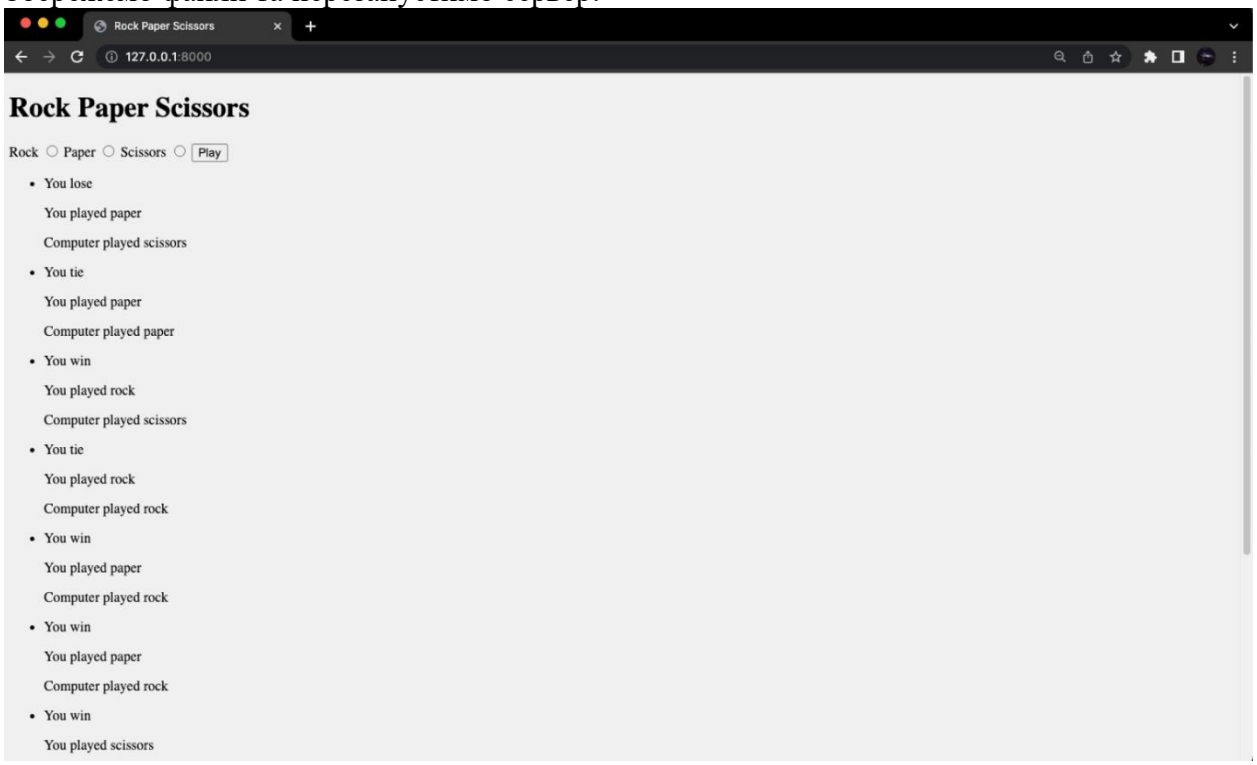
Тепер потрібно виправити файл game.blade.php і відобразити ці ігри:

```
<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>
```

Цей код просто створює ненумерований список, який розташовується після форми. У списку ми додаємо елементи <li> через цикл @foreach . У ньому ми проходимо по всіх наших іграх і виводимо для кожної гри результат, хід гравця та комп'ютера.

Збережемо файли та перезапустимо сервер:



Ура! Ми виводимо результати завершених ігор, а нові ігри успішно зберігаються.

Прикрашаємо гру за допомогою CSS

Залишилося трохи прикрасити сайт за допомогою CSS, а потім насолодитися його красою.

Відкриємо папку public і створимо файл styles.css . Додамо до нього наступний код:

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

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

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

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

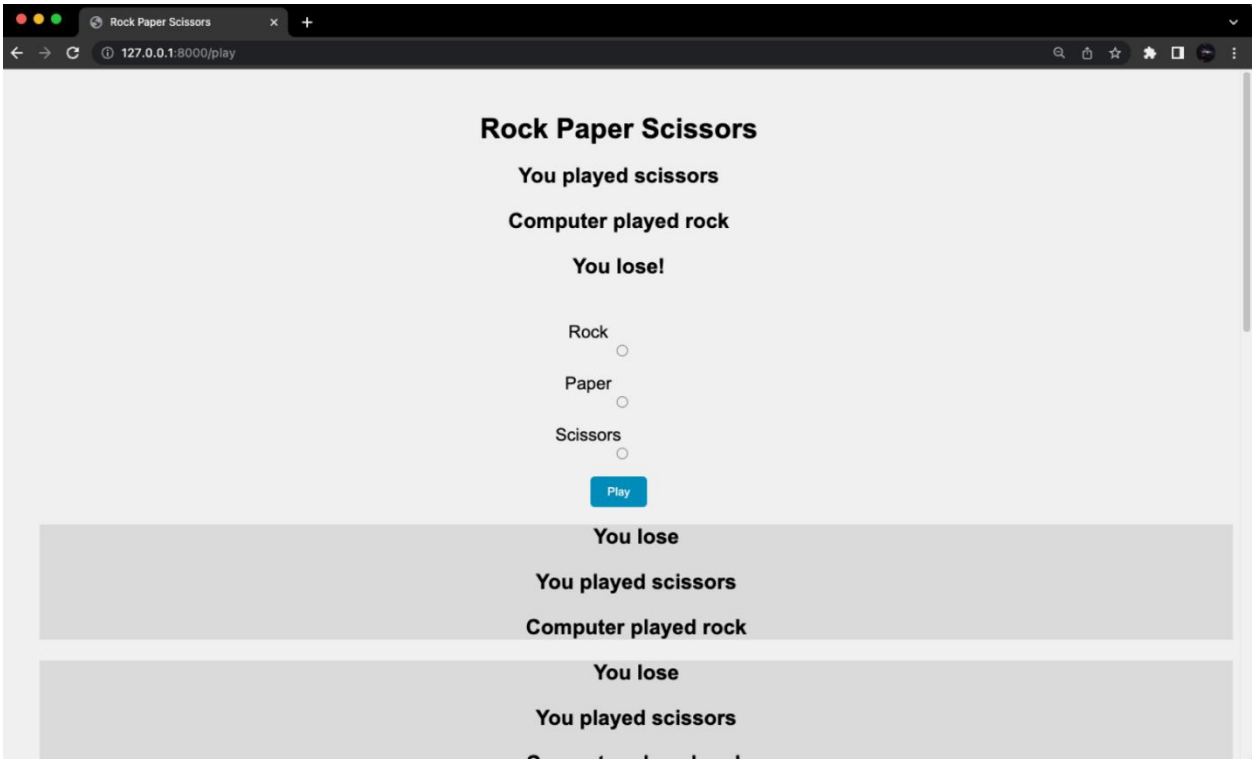
```
input[type="radio"] {
    margin-left: 10px;
    padding-bottom: 50px;
}

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;
}

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

ul {
    list-style-type: none;
}

li {
    background: #dadada;
}
```

Тут ми трохи зачесали всі елементи сторінки, щоб вони відображалися трохи симпатичнішими. Наприклад, у тезі body ми задали шрифт та колір фону, а для всіх параграфів p задали розмір шрифту, зробили його жирним та вирівняли текст по центру.

Тепер збережемо CSS-файл і відкриємо файл game.blade.php , щоб підключити до нього нові стилі:

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

Вийшло приблизно так:



Не найкращий дизайн, але концепцію ви, сподіваюся, зрозуміли. Можете спробувати самостійно довести стилі до досконалості та відправити гру на якусь премію в галузі веб-дизайну.

А ось і повний код усіх файлів, які ми створювали:

web.php:

GameController.php:

game.blade.php:

styles.css:

_create_game_statistics_table.php:

```php
<?php

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

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        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();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
```

```
    {
        Schema::dropIfExists('game_statistics');
    }
};
```