Центросоюз РФ

Нижегородский областной союз потребительских обществ

Нижегородский экономико-технологический колледж

ЗАДАНИЕ НА ПРАКТИКУ

УП.02 ОСУЩЕСТВЛЕНИЕ ИНТЕГРАЦИИ ПРОГРАММНЫХ МОДУЛЕЙ

Ананьев Артем Алексеевич

Фамилия, имя, отчество обучающегося

Специальность*09.02.07 Информационные системы и программирование* ..

Код и наименование специальности

*3* курс *34П* группа

Место прохождения практики:

*Лаборатория программного обеспечения*

*и сопровождения компьютерных систем*

(Название лаборатории, кабинета)

Сроки практики с *11 ноября* 2024 г. по *15 ноября* 2024 г.

Всего часов по практике *36* час.

|  |  |  |  |
| --- | --- | --- | --- |
| **№ п/п** | **Задание, раскрывающее содержание практики** | **Даты выполнения задания** | **Форма отчетности** |
| 1 | Изучение требований и архитектуры проекта веб-приложения | 11.11.2024 | Описание выполненной работы |
| 2 | Разработка веб-приложения клиент-серверной архитектуры с применением фреймворка Laravel. | 12.11.2024 | Описание выполненной работы, листинги кода, скриншоты браузера |
| 3 | Разработка API. Создание миграций, моделей, маршрутов и контроллеров. Тестирование с помощью инструмента Postman | 13.11.2024 | Описание выполненной работы, листинг добавленного php-кода, скриншоты Postman |
| 4 | Организация взаимодействия с базой данных. Реализация CRUD-операций. Тестирование | 13.11.2024 | Описание выполненной работы, листинг добавленного php-кода, скриншоты Postman |
| 5 | Тестирование и отладка кода, тестирование компонентов, интеграционное тестирование | 14.11.2024 | Описание выполненной работы, скриншоты Postman и браузера |
| 6 | Оформление отчетной документации по практике | 15.11.2024 | Задание на практику, дневник, отчет, аттестационный лист |
| 7 | Дифференцированный зачет | 15.11.2024 | Защита дневника-отчета |

Руководитель практики *преподаватель* . \_\_\_\_\_\_\_\_\_\_\_\_\_ *Кисарова М.В.* .

должность подпись ФИО

Арзамас, 2024

Центросоюз РФ

Нижегородский областной союз потребительских обществ

Нижегородский экономико-технологический колледж

ДНЕВНИК учебной ПРАКТИКИ

*ПМ.02 Осуществление интеграции программных модулей*

Индекс и название профессионального модуля

Специальность

*09.02.07 Информационные системы и программирование*

Код и наименование специальности

Обучающегося(йся) *3* курса *34П* группы

Ананьев Артем Алексеевич

Фамилия, имя, отчество обучающегося

Место практики: *Нижегородский экономико-технологический колледж*

*Лаборатория программного обеспечения*

*и сопровождения компьютерных систем*

Название организации, лаборатории, кабинета

Сроки практики с *11 ноября* 2024 г. по *15 ноября* 2024 г.

Всего часов по практике *36* час.

Руководитель практики *преподаватель* \_\_\_\_\_\_\_\_\_\_\_\_\_\_ *Кисарова М.В.*

должность подпись ФИО

Зав.производственной практикой \_\_\_\_\_\_\_\_\_\_\_\_\_\_ *Кияшина С.В.*

подпись ФИО

Арзамас, 2024

СОДЕРЖАНИЕ ДНЕВНИКА

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Описание выполненной работы | Кол-во  часов | Подпись руководителя практики |
| 1 | 2 | 3 | 4 |
| 11.11.2024 | Изучение требований и | 2 |  |
| 11.11.2024 | архитектуры проекта | 2 |
| 11.11.2024 | веб-приложения | 2 |
| 12.11.2024 | Разработка веб-приложения | 2 |  |
| 12.11.2024 | в архитектуре MVC | 2 |
| 12.11.2024 | с применением фреймворка Laravel. | 2 |
| 13.11.2024 | Настройка конфигурации приложения | 2 |  |
| 13.11.2024 | Организация связи с базой данных | 2 |
| 13.11.2024 | Разработка миграций, создание моделей | 2 |
| 13.11.2024 | Тестирование взаимодействия с базой данных | 2 |  |
| 14.11.2024 | Создание API-маршрутов согласно ресурсам | 2 |
| 14.11.2024 | Разработка контроллеров с методами | 2 |
| 14.11.2024 | index, show, store, update, destroy | 2 |  |
| 14.11.2024 | Тестирование с помощью инструмента Postman | 2 |
| 15.11.2024 | Реализация CRUD-операций | 2 |
| 15.11.2024 | Create, Read, Delete, Update | 2 |  |
| 15.11.2024 | Формирование отчетной документации | 2 |
| 15.11.2024 | Защита дневника-отчета по учебной практике | 2 |

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_ / Ананьев А.А. /

Руководитель практики \_\_\_\_\_\_\_\_\_\_\_\_\_ / *Кисарова М.В.* /

подпись Ф.И.О.

Центросоюз РФ

Нижегородский областной союз потребительских обществ

Нижегородский экономико-технологический колледж

ОТЧЕТ О прохождении учебной ПРАКТИКИ

*ПМ.02 Осуществление интеграции программных модулей*

Индекс и название профессионального модуля

Специальность *09.02.07 Информационные системы и программирование*

Код и наименование специальности

Обучающегося(йся) *3* курса *34П* группы

Ананьев Артем Алексеевич

Фамилия, имя, отчество обучающегося

Место практики: *Нижегородский экономико-технологический колледж*

*Лаборатория программного обеспечения*

*и сопровождения компьютерных систем*

Название организации, лаборатории, кабинета

Сроки практики с *11 ноября* 2024 г. по *15 ноября* 2024 г.

Всего часов по практике *36* час.

Руководитель практики *преподаватель* \_\_\_\_\_\_\_\_\_\_\_\_\_\_ *Кисарова М.В.*

должность подпись ФИО

Зав.производственной практикой \_\_\_\_\_\_\_\_\_\_\_\_\_\_ *Кияшина С.В.*

подпись ФИО

Оценка по практике

Арзамас, 2024

СОДЕРЖАНИЕ ОТЧЕТА

1. Разработка игры «Камень-ножницы-бумага» с применением фреймворка

Laravel 3

1. Разработка миграций 7
2. Разработка API-маршрутов и контроллеров 8
3. Реализация CRUD-операций 10
4. Тестирование с помощью инструмента Postman 12

Выводы 13

1.РАЗРАБОТКА ИГРЫ «КАМЕНЬ-НОЖНИЦЫ-БУМАГА» С ПРИМЕНЕНИМ ФРЕЙМВОРКА LARAVEL

Laravel – это бесплатный PHP-фреймворк с открытым исходным кодом. Laravel разработали в качестве помощника при создании сложных веб-ресурсов и приложений. С его помощью специалисты упрощают процесс аутентификации, а также работу с БД, кэширование, сессии, структуру приложения, маршрутизацию и другие не менее важные процессы.

Возможностей у платформы Laravel немало. Одна из них – построение логичной архитектуры для проектов любой сложности и типа. Платформа характеризуется:

-высокой производительностью;

-возможностью интеграции с другими платформами, а также библиотеками;

-немалым количеством интересных возможностей для разработчиков сайтов и приложений.

Создание простой игры «камень-ножницы-бумага».

Для начала мы создали новый проект с помощью команды, введенной в командною строку (composer create-project --prefer-dist laravel/laravel rock-paper-scissors). После этого следует организовать базу данных, чтобы сохранять статистику игры. Для этого мы создали в phpMyAdmin базу данных и подключили ее в файле .env. Далее мы приступили к созданию миграций. В терминале выполняем команду php artisan make:migration create\_game\_statistics\_table и после задаем структуру таблицы (листинг 1).

Листинг 1 – Структура таблицы

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();

});

}

Запускаем миграцию командой php artisan migrate.

Нам понадобятся два маршрута: корневой / и игровой /play. Поэтому мы добавляем маршруты в файл web.php.

Листинг 2 - Маршруты

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

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

В начало php-сценария необходимо добавить ссылку на папку с контроллерами: namespace App\Http\Controllers;.

Создаем контроллер с помощью команды php artisan make:controller GameController и вносим в него код описывающий основную логику игры.

Листинг 3 – Код игры

class GameController extends Controller

{

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');

$computerChoice = rand(1,3);

if ($computerChoice == 1) {

$computerChoice = 'rock';

}

elseif ($computerChoice == 2)

{

$computerChoice = 'paper';

}

else {

$computerChoice = 'scissors';

}

if ($playerChoice == $computerChoice) {

$result = 'tie';

}

elseif (($playerChoice == 'rock' && $computerChoice == 'scissors') || ($playerChoice == 'paper' && $computerChoice == 'rock') || ($playerChoice == 'scissors' && $computerChoice == 'paper' )) {

$result = 'win';

}

else {

$result = 'lose';

}

DB::table('game\_statistics')->insert([

'player\_name' => 'Player 1',

'computer\_choice' => $computerChoice,

'player\_choice' => $playerChoice,

'result' => $result,

'created\_at' => now(),

'updated\_at' => now()

]);

return view ('game', ['result' => $result, 'player\_choice' => $playerChoice, 'computer\_Choice'=> $computerChoice, 'last\_games'=>$last\_games]);

}

}

Метод index() из файла GameController.php отображает страницу с игрой, но пока на ней ничего нет, поэтому нужно добавить пару кнопок. Открываем папку resources/views и создаём там файл с названием game.blade.php. Добавляем туда следующий код, чтобы создать игровой интерфейс:

Листинг 4 – Код интерфейса игры

<h1>Камень Ножницы Бумага</h1>

@if(isset($result))

<p>You played {{$player\_choice}}</p>

<p>Computer played {{$computer\_Choice}}</p>

<p> You {{ $result }}!</p>

@endif

<form method="POST" action="{{ route('game.play') }}">

@csrf

<label for="rock">Камень</label>

<input type="radio" name="choice" id="rock" value="rock">

<label for="paper">Бумага</label>

<input type="radio" name="choice" id="paper" value="paper">

<label for="scissors">Ножницы</label>

<input type="radio" name="choice" id="scissors" value="scissors">

<button type="submit">Играть</button>

</form>

<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>

Осталось только оформить нашу игру с помощью CSS-стилей. Открываем папку public и создаем файл styles.css. Добавим в него следующий код:

Листинг 5 – CSS-стили

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 0.3 ease;

}

p {

text-align: center;

margin-top: 20px;

font-size: 24px;

font-weight: bold;

}

ul {

list-style-type: none;

}

li {

background: #dadada;

}

После в файле game.blade.php подключаем новые стили.

Листинг 6 – Подключение стилей

<link rel="stylesheet" href="{{asset('styles.css')}}">

Игра готова! Frontend-часть приложения можно увидеть на рисунке 1.

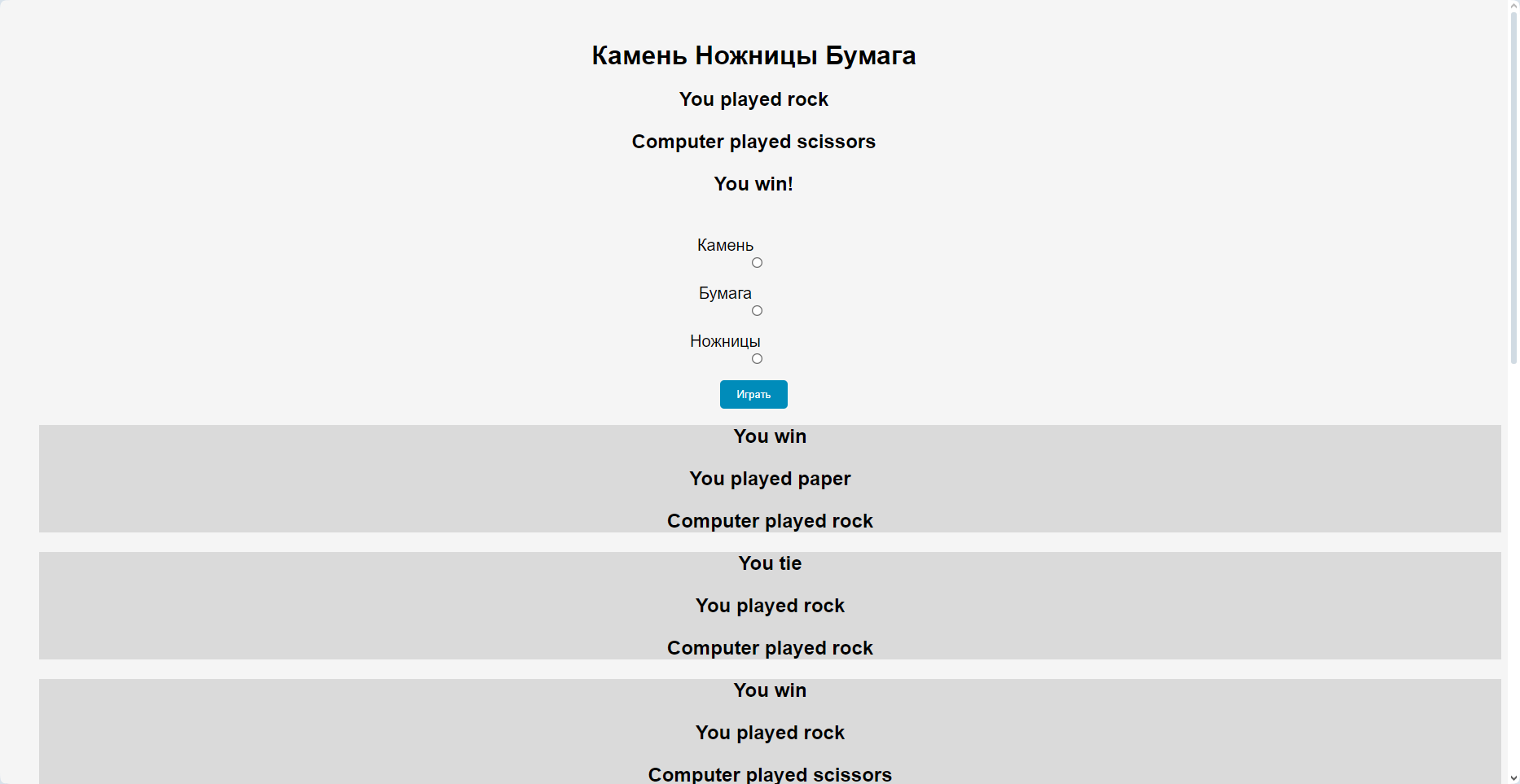


Рисунок 1 – Готовая игра «Камень ножницы бумага»

Вывод по параграфу.

2.РАЗРАБОТКА МИГРАЦИЙ

Миграции в Laravel – способ управления изменениями в структуре базы данных вашего проекта. Они позволяют создавать и изменять таблицы, индексы, ограничения и другие элементы базы данных с помощью кода, а не с использованием SQL-запросов.

Миграции позволяют легко развертывать и поддерживать структуру базы данных, а также включать изменения в распределение вашего проекта с помощью системы контроля версий.

Чтобы создать новую миграцию нужно использовать Artisan команду php artisan make:migration [имя\_миграции]. Миграция будет создана в папке database/migrations. Файл миграции будет содержать две функции:

-up(): содержит код, который будет выполняться при применении миграции.

-down(): содержит код, который будет выполняться при откате миграции.

Далее необходимо использовать Artisan команду php artisan migrate для применения миграции.

Таким способом мы создали две миграции.

Листинг 7 – Примеры миграций

public function up(): void

{

Schema::create(‘categories’, function (Blueprint $table) {

$table->id();

$table->string(‘name’);

$table->timestamps();

});

}

public function down(): void

{

Schema::dropIfExists(‘categories’);

}

public function up(): void

{

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

$table->id();

$table->string(‘name’);

$table->foreignId(‘category\_id’)->constrained(‘categories’)->cascadeOnDelete();

$table->foreignId(‘user\_id’)->constrained(‘users’)->cascadeOnDelete();

$table->timestamps();

});

}

public function down(): void

{

Schema::dropIfExists(‘posts’);

}

Вывод по параграфу.

3.РАЗРАБОТКА API-МАРШРУТОВ И КОНТРОЛЛЕРОВ

Разработка API маршрутов и контроллеров – это ключевой аспект создания веб-приложений, которые предоставляют доступ к своим данным и функционалу через API (Application Programming Interface). Этот процесс включает в себя определение конечных точек (endpoints), обработку запросов и формирование ответов.

Маршруты определяют, какие URL-адреса соответствуют каким функциям в вашем приложении. В Laravel это обычно делается в файле routes/api.php.

Вот пример маршрутов которые мы сделали:

Листинг 8 – Примеры маршрутов

Route::post('/users/{users}', [UsersController::class, 'store']);

Route::get('/users', [UsersController::class, 'index']);

Route::get('/users/{users}', [UsersController::class, 'show']);

Route::post('/tags/{tags}', [TagController::class, 'store']);

Route::get('/tags', [TagController::class, 'index']);

Route::put('/tags/{tags}', [TagController::class, 'update']);

Route::delete('/tags/{tags}', [TagController::class, 'destroy']);

Route::post('/ads/{ads}', [AdsController::class, 'store']);

Route::get('/ads', [AdsController::class, 'index']);

Route::get('/ads/{ads}', [AdsController::class, 'show']);

Route::put('/ads/{ads}', [AdsController::class, 'update']);

Route::delete('/ads/{ads}', [AdsController::class, 'destroy']);

Route::post('/applications/{applications}', [ApplicationController::class, 'store']);

Route::get('/messages', [MessageController::class, 'index']);

Route::post('/messages/{messages}', [MessageController::class, 'store']);

Контроллер (Controller) в Laravel — это класс, который отвечает за обработку HTTP-запросов и формирование HTTP-ответов. Он служит посредником между вашим приложением и внешним миром.

Основные задачи контроллера:

Приём запросов: Контроллер получает HTTP-запросы (GET, POST, PUT, PATCH, DELETE), которые приходят от браузера или других приложений.

Валидация данных: Контроллер проверяет данные, полученные от клиента, на соответствие заданным правилам.

Логика приложения: Контроллер выполняет бизнес-логику вашего приложения:

* Доступ к базе данных
* Обработка данных
* Выполнение операций с файлами
* Вызов других сервисов

Формирование ответа: Контроллер формирует HTTP-ответ, который будет отправлен клиенту:

* HTML-код для веб-страниц
* JSON для API
* XML или другие форматы данных
* Перенаправление на другую страницу
* Ошибка 404, 500 и т.д.

**Листинг 9 – Пример контроллера**

<?php

namespace App\Http\Controllers;

use App\Models\Ads;

use Illuminate\Http\Request;

class AdsController extends Controller

{

public function index()

{

}

public function store(Request $request)

{

}

public function show(Ads $ads)

{

}

public function update(Ads $ads, Request $request)

{

}

public function destroy(Ads $ads)

{

}

}

4.РЕАЛИЗАЦИЯ CRUD-ОПЕРАЦИЙ

**CRUD** - это **четыре основные операции, выполняемые при управлении данными и манипулировании ими в рамках разработки программного обеспечения**.

**Операция CREATE** добавляет новые записи данных или объекты в систему хранения данных, такую как база данных или API.

**Операция READ** необходима для получения и отображения данных из системы хранения, такой как база данных или API.

**Операция UPDATE** позволяет пользователям изменять ранее сохранённую информацию в приложении. **Операция DELETE** отвечает за удаление данных из системы хранения, такой как база данных или API.

Листинг 10 – Примеры использования CRUD-операций

public function index()

{

$ads = Ads::all();

return $ads;

}

public function store(Request $request)

{

return Ads::create($request->all());

}

public function show(Ads $ads)

{

return $ads;

}

public function update(Ads $ads, Request $request)

{

$ads->update($request->all());

$ads->save();

return $ads;

}

public function destroy(Ads $ads)

{

$ads->delete();

return ["message" => "$ads->name Deleted!"];

}

}

public function index()

{

$ads = Ads::all();

return $ads;

}

public function store(Request $request)

{

return Ads::create($request->all());

}

public function show(Ads $ads)

{

return $ads;

}

public function update(Ads $ads, Request $request)

{

$ads->update($request->all());

$ads->save();

return $ads;

}

public function destroy(Ads $ads)

{

$ads->delete();

return ["message" => "$ads->name Deleted!"];

}

}

CRUD операции (Create, Read, Update, Delete) - это базовые действия, которые необходимы для управления данными в любом приложении. Они позволяют создавать новые данные, просматривать существующие, обновлять их и удалять при необходимости. CRUD являются фундаментом для большинства веб-приложений, обеспечивая интерактивность и возможность работы с информацией.

5.ТЕСТИРОВАНИЕ С ПОМОЩЬЮ ИНСТРУМЕНТА POSTMAN

**Postman** -это сервис для создания, тестирования, документирования, публикации и обслуживания API. Он позволяет создавать коллекции запросов к любому API, применять к ним разные окружения, настраивать мок-серверы, писать автотесты на [JavaScript](https://skillbox.ru/media/code/chto-takoe-javascript-i-zachem-on-nuzhen/?utm_source=media&utm_medium=link&utm_campaign=all_all_media_links_links_articles_all_all_skillbox), анализировать и визуализировать результаты запросов.

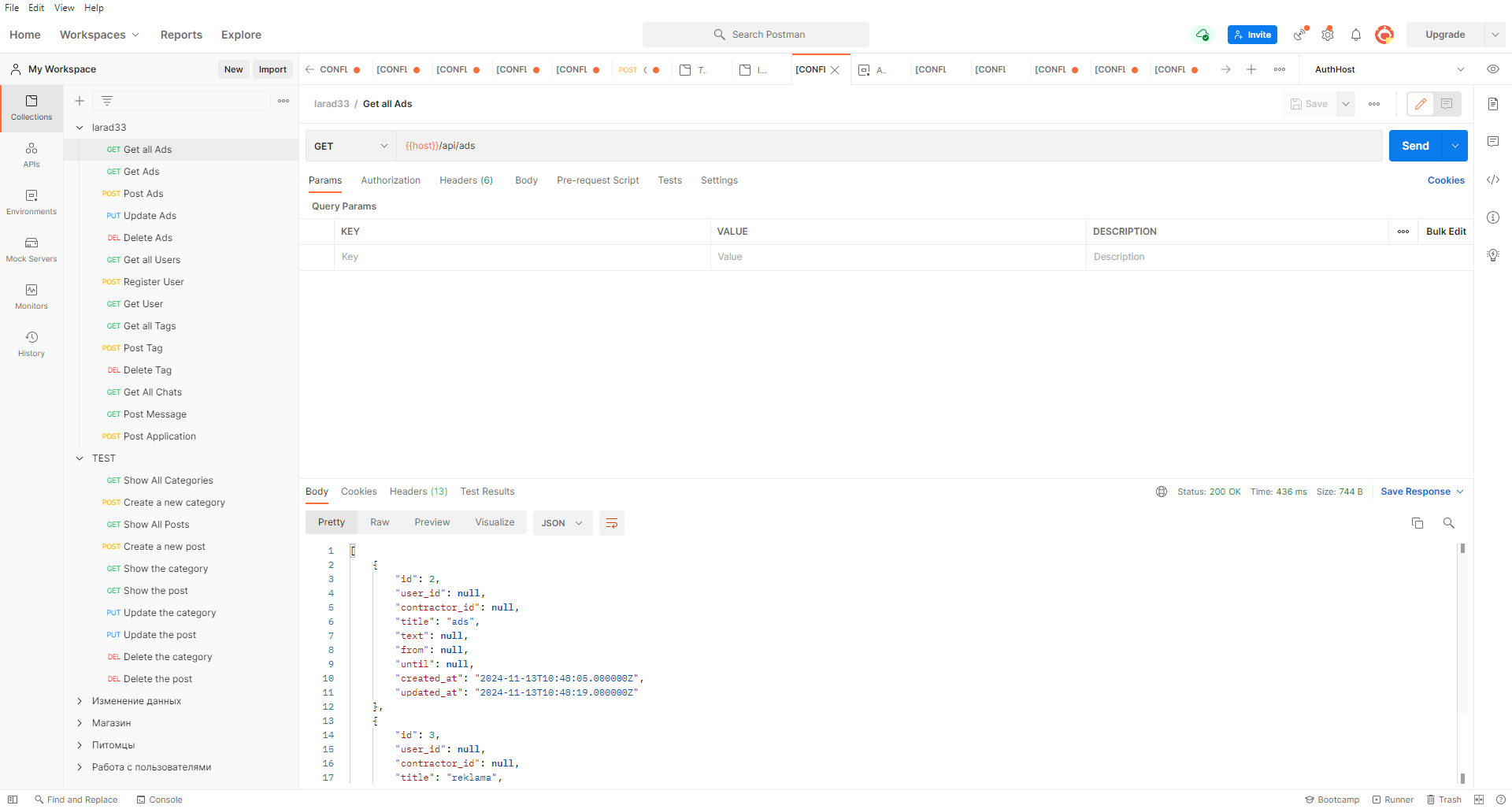


Рисунок 2 – API в Postman

Postman является незаменимым инструментом для всех, кто работает с API: разработчиков, тестировщиков, менеджеров API и других специалистов. Он позволяет упростить разработку, тестирование и документирование API и сделать их более качественными и надежными Postman — это популярный инструмент для тестирования API, который предоставляет широкий спектр возможностей для разработчиков и тестировщиков. Он позволяет легко отправлять HTTP-запросы, анализировать ответы, автоматизировать тестирование и многое другое. В этом докладе мы рассмотрим основные аспекты использования Postman для тестирования, включая примеры и рекомендации. Postman позволяет отправлять различные типы HTTP-запросов (GET, POST, PUT, DELETE и другие). Это делает его идеальным инструментом для тестирования RESTful API. Пользовательский интерфейс интуитивно понятен и удобен для создания и редактирования запросов.Коллекции позволяют организовывать запросы в группы для удобного управления и повторного использования. Окружения помогают задавать разные наборы параметров для разных этапов разработки (разработка, тестирование, производство).

Postman поддерживает написание тестов на JavaScript, что позволяет проверять корректность ответов от сервера. Также существует возможность автоматической генерации тестов на основе примеров ответов. Для автоматизации тестирования можно использовать функцию Collections Runner.

Postman предоставляет инструменты для мониторинга API, позволяя настроить регулярные проверки работоспособности сервисов. Кроме того, можно автоматически генерировать документацию на основе коллекций, что облегчает понимание и поддержку API.

Postman является мощным и удобным инструментом для тестирования API. Его функциональность охватывает широкий спектр задач, начиная от простого ручного тестирования до автоматизации и мониторинга. Использование Postman помогает улучшить качество разрабатываемых приложений и ускорить процессы тестирования.

ВЫВОДЫ

В ходе работы над проектом по интеграции программных модулей, используя Laravel и Postman, были получены практические навыки и углублены теоретические знания в области разработки API-интерфейсов и их взаимодействия.Эффективность Laravel для разработки API: Laravel предоставил удобный инструмент для разработки серверной части API, обеспечивая структурированное управление данными, маршрутизацией и контроллерами. Использование моделей и репозиториев позволило разделить логику и код, что облегчило поддержание и расширение API.

Роль Postman в тестировании и разработке API: Postman стал незаменимым инструментом для тестирования и отладки API, позволив визуализировать запросы, ответы, и проверять работоспособность на разных этапах разработки. Он значительно ускорил процесс тестирования и обеспечил более качественный API.

Важность четкого проектирования API: Правильное планирование интерфейсов (endpoints), структура данных (форматы запросов и ответов), и использование HTTP-методов были crucial для успешной интеграции. Четкое определение форматов обмена данными и валидации данных на обеих сторонах интеграции было решающим фактором для исключения проблем.

Практическое применение принципов модульности: Реализация интеграции требовала четкого разделения ответственности между модулями. Интеграция модулей с помощью четких API-контрактов повысила гибкость и масштабируемость всей системы.

Значение тестирования и отладки: Процесс интеграции включал в себя множество этапов тестирования. Postman использовался для отладки запросов и ответов на каждом этапе, что помогло выявить и исправить ошибки до этапа запуска.

Преимущества использования REST API: REST API-архитектура продемонстрировала свою эффективность, обеспечивая простоту и гибкость для взаимодействия между модулями. Отсутствие необходимости в специальном протоколе взаимодействия сделало процесс интеграции более плавным.

**Аттестационный лист**

Обучающийся Ананьев Артем Алексеевич

ФИО

*3* курса по специальности *09.02.07 Информационные системы и программирование*

код и наименование специальности

прошел(ла) учебную практику по профессиональному модулю

*ПМ.02. Осуществление интеграции программных модулей*

индекс и название профессионального модуля

в объеме *36* часов с « *11* »  *ноября*  2024 г. по « *15* »  *ноября*  2024 г.

*в частном профессиональном образовательном учреждении*

*«Нижегородский экономико-технологический колледж»*

\_\_ *в лаборатории программного обеспечения и сопровождения компьютерных систем*

название лаборатории, кабинета

Критерии оценки:

5 баллов – качество выполненных заданий высокое, ярко выраженное, проявляется всегда;

4 балла – качество выполненных заданий среднее, проявляется в зависимости от ситуации;

3 балла – качество выполненных заданий умеренно выраженное, проявляется редко;

2 балла – качество выполненных заданий выражено слабо, не выражено.

**Виды и качество выполнения работ в период учебной практики**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Умения, формируемые в составе профессиональных компетенций** | **Виды работ,**  **выполненных обучающимся во время практики** | **Объем работ, час.** | **Оценка** | |
| ПК 2.1, 2.2:   * разработка требований к программным модулям для интеграции их в программные средства на базе имеющейся архитектуры и автоматизации бизнес-процессов; * организация постобработки данных; | Изучение требований и архитектуры клиент-серверного приложения | 6 |  | |
| Организация взаимодействия с базой данных в приложении | 12 |
| ПК 2.3, 2.4, 2.5:   * использовать методы для получения кода с заданной функциональностью и степенью качества; | Разработка и тестирование веб-приложения клиент-серверной архитектуры с использованием фреймворка Laravel и инструмента Postman | 12 | |  |
| Оформление отчетной документации по практике | 6 | |

Программа учебной практики \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

освоена / не освоена

Итоговая оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель практики \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ *Кисарова М.В.*

« *15* »  *ноября*  2024 г.

Зав. производственной практикой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ *Кияшина С.В.*

Ознакомлен: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ *Ананьев А.А.*

« *15* »  *ноября*  2024 г.