|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №4**

по дисциплине «Разработка серверных частей интернет-ресурсов»

**Тема практической работы:** Реализация взаимодействия клиента и сервера с использованием технологии API

**Студент группы** ИКБО-16-20 Косогоров Кирилл Станиславович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

**Руководитель практической работы** преподаватель Волков М.Ю.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Работа представлена «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Москва 2022

Оглавление

[1. Цель работы 3](#_Toc125999917)

[2. Постановка задачи 3](#_Toc125999918)

[3. Ход работы 3](#_Toc125999919)

[4. Спецификация API 10](#_Toc125999920)

[5. Вывод 15](#_Toc125999921)

[6. Ответы на вопросы к практической работе 16](#_Toc125999922)

[7. Список использованной литературы 22](#_Toc125999923)

1. Цель работы

Предполагается реализация интерфейса прикладного программирования для доступа к некоторым данным по варианту. Предполагается реализация серверной части обработки запросов и тестирование данного интерфейса с использованием программы Postman. Для реализации данного сервиса предлагается использовать серверную конфигурацию, модернизированную в течение первых трех практических работ. Важной частью данной практической работы является сохранение функциональности реализованной в практической работе No3. То есть интерфейс предлагается создать уже в существующем веб-приложении. Также предполагается использование темы практической работы No3 для продолжения модернизирования собственной системы. Изменение темы согласовывается отдельно с преподавателем. Хранение данных предполагается уже в существующей базе данных.

1. Постановка задачи

Технические требования к реализации интерфейса:

1. Доступ как минимум к 2 независимым сущностям.

2. Реализация как минимум операций группы CRUD(создание, чтение, обновление, удаление). Приветствуется реализация дополнительной функциональности.

3. Тестирование всех функциональных возможностей созданного интерфейса с использованием программы Postman.

1. Ход работы

К данной практической работе была создана структура контейнеров (Листинг 1).

Листинг 1. Структура docker-compose.yml

version: '3'

services:

  apache:

    build: ./php-apache

    container\_name: APACHE

    volumes:

      - ./src:/var/www/apache-server/html

      - ./ApacheServerConf/apache2.conf:/etc/apache2/sites-enabled/000-default.conf

    depends\_on:

      mysql:

        condition: service\_healthy

  nginx:

    image: nginx:latest

    container\_name: NGINX

    ports:

      - 80:80

    volumes:

      - ./src:/var/www/apache-server/html

      - ./NginxServerConf/nginx.conf:/etc/nginx/nginx.conf

    depends\_on:

      - apache

  mysql:

    container\_name: MYSQL

    hostname: MYSQL

    image: mysql

    environment:

      MYSQL\_ROOT\_PASSWORD: password

      HOSTNAME: MYSQL

    volumes:

      - "./sql:/docker-entrypoint-initdb.d"

      - "dbdata:/var/lib/mysql"

    healthcheck:

      test: [ "CMD", "mysqladmin" ,"ping", "-h", "localhost" ]

      timeout: 1s

      retries: 10

volumes:

  dbdata:

Запуск проекта осуществляется в несколько шагов. Сначала запускается база данных MYSQL и SQL-скрипт, который создает пользователя и схему с таблицей, в которой будут храниться данные пользователей. MYSQL-скрипт приведен на листинге 2.

Листинг 2. Файл init.sql

CREATE DATABASE IF NOT EXISTS appDB;

CREATE USER IF NOT EXISTS 'user'@'%' IDENTIFIED WITH mysql\_native\_password BY 'password';

GRANT SELECT,UPDATE,INSERT,DELETE ON appDB.\* TO 'user'@'%';

FLUSH PRIVILEGES;

USE appDB;

CREATE TABLE IF NOT EXISTS users (

    ID INT(11) NOT NULL AUTO\_INCREMENT,

    username VARCHAR(32) NOT NULL,

    password VARCHAR(256) NOT NULL,

    email VARCHAR(64) NOT NULL,

    PRIMARY KEY (ID)

);

CREATE TABLE IF NOT EXISTS `painting`

(

    ID          INT(11)      NOT NULL AUTO\_INCREMENT,

    name        VARCHAR(255)  NOT NULL,

    description VARCHAR(256) NOT NULL,

    price       INT(7)       NOT NULL,

    PRIMARY KEY (ID)

);

INSERT INTO users (username, password, email) VALUES ("kirill", "{SHA}QL0AFWMIX8NRZTKeof9cXsvbvu8=", "kosgor2001@yandex.ru");

INSERT INTO `painting` (name,description,price) VALUES ('Wetsuit Aqualung Diveflex','One-piece suit made from soft and stretchy 5.5mm UltraStretch neoprene with back zip. Fits great, easy to put on and take off. The wrists and ankles are equipped with 2.5 mm smooth skin seals without zippers to prevent water from entering the suit.

',23400);

INSERT INTO `painting` (name,description,price) VALUES ('Compensator vest Prodive PRO 1000 den','Reliable compensator BCD Prodive PRO 1000 den from the manufacturer Prodive is suitable for beginner divers, as it is located in the budget price category. The compensator is made of durable wear-resistant material Cordura 1000 den. And the soft back gives comfort and relieves the load from the lower back when worn with a fixed balloon.

',36000);

INSERT INTO `painting` (name,description,price) VALUES ('Neoprene gloves 2,5 mm, kevlar','Problue neoprene gloves are made from 2.5mm thick neoprene and have a Kevlar coating on the palms and fingers. Kevral gives excellent wear resistance to the product. Problue neoprene gloves, in addition to excellent performance, also have a democratic cost.

',2450);

Далее запускается образ php-apache на основе 3 практики. Был написан REST API для сущностей users и paintings (Листинги 3-4).

*Листинг 3 – Файл user.php*

<?php

$requestMethod = $\_SERVER["REQUEST\_METHOD"];

header('Content-Type: application/json');

$con = new mysqli("MYSQL", "user", "password", "appDB");

$answer = array();

switch ($requestMethod) {

    case 'GET':

        if (empty(isset($\_GET['id']))) {

            $result = $con->query("SELECT \* FROM users;");

            while ($row = $result->fetch\_assoc()) {

                $answer[] = $row;

            }

        } else {

            $query\_result = $con->query("SELECT \* FROM users WHERE ID = " . $\_GET['id'] . ";");

            $result = $query\_result->fetch\_row();

            $answer = $result;

        }

        if (!empty($result)) {

            http\_response\_code(200);

            echo json\_encode($answer);

        } else {

            http\_response\_code(204);

        }

        break;

    case 'POST':

        $json = file\_get\_contents('php://input');

        $user = json\_decode($json);

        if (!empty($user->{'username'}) && !empty($user->{'password'}) && !empty($user->{'email'})) {

            $username = $user->{'username'};

            $password = $user->{'password'};

            $email = $user->{'email'};

            $query\_result = $con->query("SELECT \* FROM users WHERE username='" . $username . "'");

            if (!empty($result)) {

                http\_response\_code(409);

            } else {

                $password = crypt($password);

                $stmt = $con->prepare("INSERT INTO users (username, password, email) VALUES (?, ?, ?)");

                $stmt->bind\_param('sss', $username, $password, $email);

                $stmt->execute();

                http\_response\_code(201);

            }

        } else {

            http\_response\_code(422);

        }

        break;

    case 'PUT':

        $json = file\_get\_contents('php://input');

        $user = json\_decode($json);

        if (!empty($user->{'username'}) && !empty($user->{'email'})) {

            if (empty(isset($\_GET['id']))) {

                http\_response\_code(422);

            } else {

                $query\_result = $con->query("SELECT \* FROM users WHERE ID='" . $\_GET['id'] . "'");

                $result = $query\_result->fetch\_row();

                if (!empty($result)) {

                    $query\_result = $con->query("SELECT \* FROM users WHERE username='" . $user->{'username'} . "' AND ID!='" . $\_GET['id'] . "'");

                    $result = $query\_result->fetch\_row();

                    if (!empty($result)) {

                        http\_response\_code(409);

                    } else {

                        $con->query("UPDATE users SET username='" . $user->{'username'} . "', email='" . $user->{'email'} . "' WHERE ID='" . $\_GET['id'] . "'");

                        http\_response\_code(200);

                    }

                } else {

                    http\_response\_code(204);

                }

            }

        } else {

            http\_response\_code(422);

        }

        break;

    case 'DELETE':

        if (empty(isset($\_GET['id']))) {

            http\_response\_code(422);

        } else {

            $query\_result = $con->query("SELECT \* FROM users WHERE ID='" . $\_GET['id'] . "'");

            $result = $query\_result->fetch\_row();

            if (!empty($result)) {

                $query\_result = $con->query("DELETE FROM users WHERE ID='" . $\_GET['id'] . "'");

                http\_response\_code(200);

            } else {

                http\_response\_code(204);

            }

        }

        break;

    default:

        http\_response\_code(405);

        break;

}

?>

*Листинг 4 – Файл painting.php*

<?php

$requestMethod = $\_SERVER["REQUEST\_METHOD"];

header('Content-Type: application/json');

$con = new mysqli("MYSQL", "user", "password", "appDB");

$answer = array();

switch ($requestMethod) {

    case 'GET':

        if (empty(isset($\_GET['id']))) {

            $result = $con->query("SELECT \* FROM painting;");

            while ($row = $result->fetch\_assoc()) {

                $answer[] = $row;

            }

        } else {

            $query\_result = $con->query("SELECT \* FROM painting WHERE ID = " . $\_GET['id'] . ";");

            $result = $query\_result->fetch\_row();

            $answer = $result;

        }

        if (!empty($result)) {

            http\_response\_code(200);

            echo json\_encode($answer);

        } else {

            http\_response\_code(204);

        }

        break;

    case 'POST':

        $json = file\_get\_contents('php://input');

        $painting = json\_decode($json);

        if (!empty($painting->{'name'}) && !empty($painting->{'description'}) && !empty($painting->{'price'})) {

            $name = $painting->{'name'};

            $description = $painting->{'description'};

            $price = $painting->{'price'};

            $query\_result = $con->query("SELECT \* FROM painting WHERE name='" . $name . "'");

            $stmt = $con->prepare("INSERT INTO painting (name, description, price) VALUES (?, ?, ?)");

            $stmt->bind\_param('sss', $name, $description, $price);

            $stmt->execute();

            http\_response\_code(201);

        } else {

            http\_response\_code(422);

        }

        break;

    case 'PUT':

        $json = file\_get\_contents('php://input');

        $painting = json\_decode($json);

        if (!empty($painting->{'name'}) && !empty($painting->{'price'})&& !empty($painting->{'description'})) {

            if (empty(isset($\_GET['id']))) {

                http\_response\_code(422);

            } else {

                $query\_result = $con->query("SELECT \* FROM painting WHERE ID='" . $\_GET['id'] . "'");

                $result = $query\_result->fetch\_row();

                if (!empty($result)) {

                    $query\_result = $con->query("SELECT \* FROM painting WHERE name='" . $painting->{'name'} . "' AND ID!='" . $\_GET['id'] . "'");

                    $result = $query\_result->fetch\_row();

                    if (!empty($result)) {

                        http\_response\_code(409);

                    } else {

                        $con->query("UPDATE painting SET name='" . $painting->{'name'} . "', price='" . $painting->{'price'} . "' WHERE ID='" . $\_GET['id'] . "'");

                        http\_response\_code(200);

                    }

                } else {

                    http\_response\_code(204);

                }

            }

        } else {

            http\_response\_code(422);

        }

        break;

    case 'DELETE':

        if (empty(isset($\_GET['id']))) {

            http\_response\_code(422);

        } else {

            $query\_result = $con->query("SELECT \* FROM painting WHERE ID='" . $\_GET['id'] . "'");

            $result = $query\_result->fetch\_row();

            if (!empty($result)) {

                $query\_result = $con->query("DELETE FROM painting WHERE ID='" . $\_GET['id'] . "'");

                http\_response\_code(200);

            } else {

                http\_response\_code(204);

            }

        }

        break;

    default:

        http\_response\_code(405);

        break;

}

?>

1. Спецификация API

* GET-запрос.

Позволяет получить информацию.

Получить информацию обо всех записях http://localhost/crud/painting.php

Получить информацию о записи с определенным id http://localhost/crud/painting.php/?id=1

Возможные коды возврата:

1. 200 – сообщает об успешном выполнении запроса
2. 204 – сообщает о том, что не существует записи

* POST-запрос.

Позволяет создать запись.

Создать запись http://localhost/crud/painting.php

Возможные коды возврата:

1. 201 – сообщает об успешном создании записи
2. 422 – сообщает о неуспешном создании записи

* PUT-запрос.

Позволяет обновить запись.

Обновить запись http://localhost/crud/painting.php?id=2

Возможные коды возврата:

1. 200 – сообщает об успешном выполнении запроса
2. 422 – сообщает о несуществовании записи с данным id
3. 409 – сообщает о том, что этот ответ отсылается, когда запрос конфликтует с текущем состоянием сервера.
4. 204 - сообщает о том, что не существует записи

* DELETE-запрос.

Позволяет удалить запись.

Удалить запись http://localhost/crud/painting.php?id=2

Возможные коды возврата:

1. 200 – сообщает об успешном выполнении запроса
2. 422 – сообщает о несуществовании записи с данным id
3. 204 – сообщает о том, что не существует записи
4. 405 – сообщает о том, что метод не разрешен.

Теперь запустим сервер и проверим работоспособность с помощью запросов.

Для начала убедимся, что скрипт создал необходимую БД (Рисунок 1).

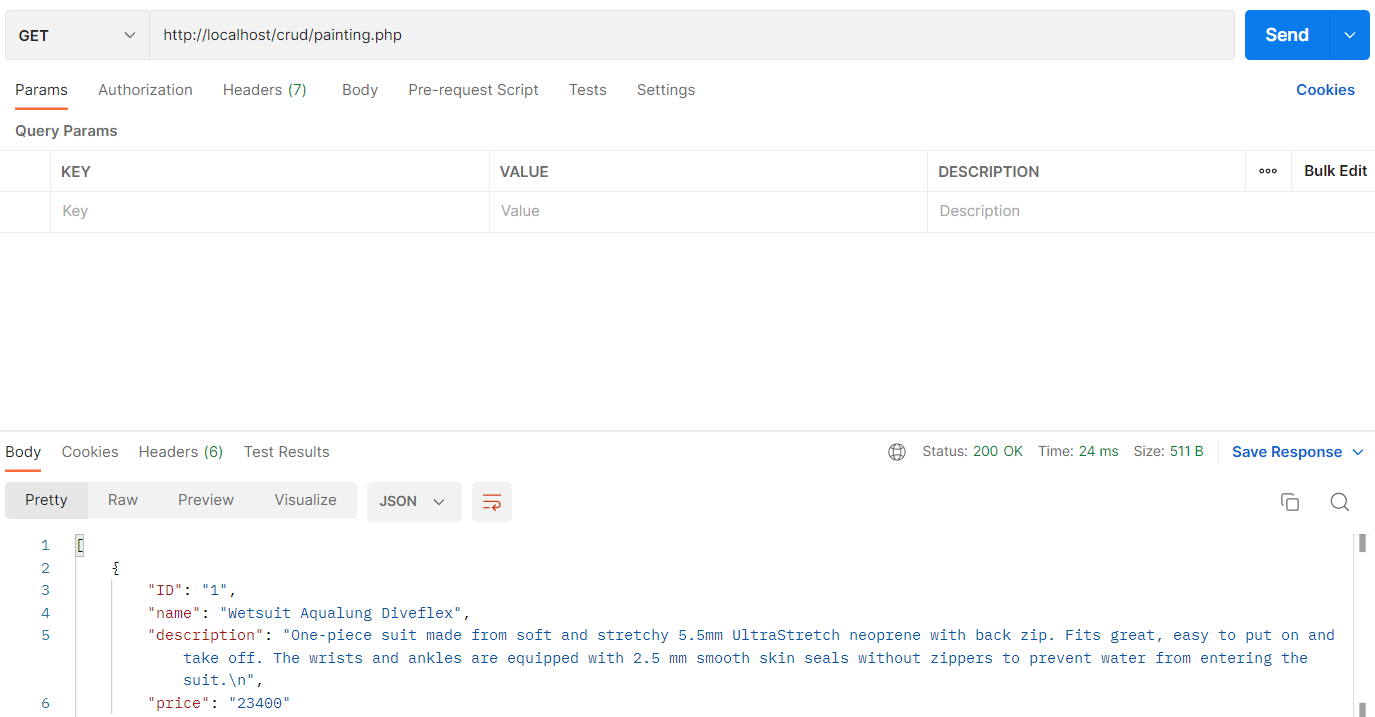


Рисунок 1 – GET-запрос

Затем попробуем создать новую запись (Рисунки 2-3).

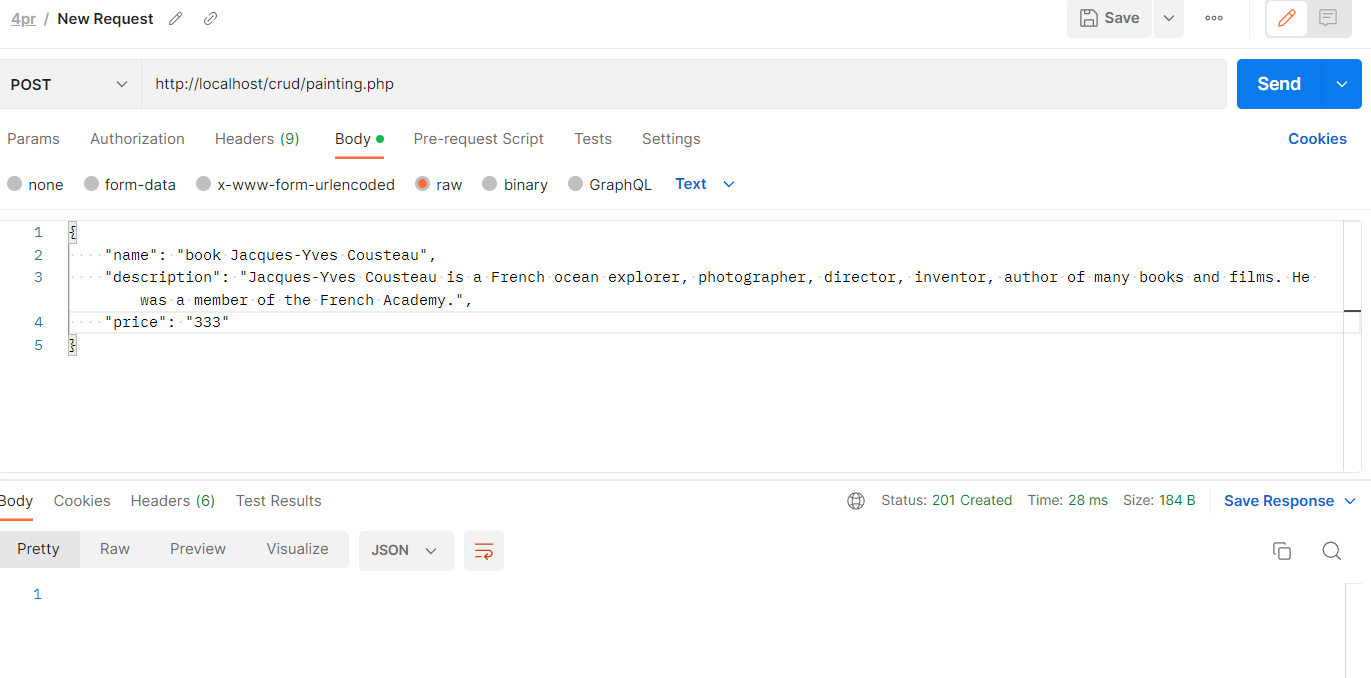


Рисунок 2 – POST-запрос

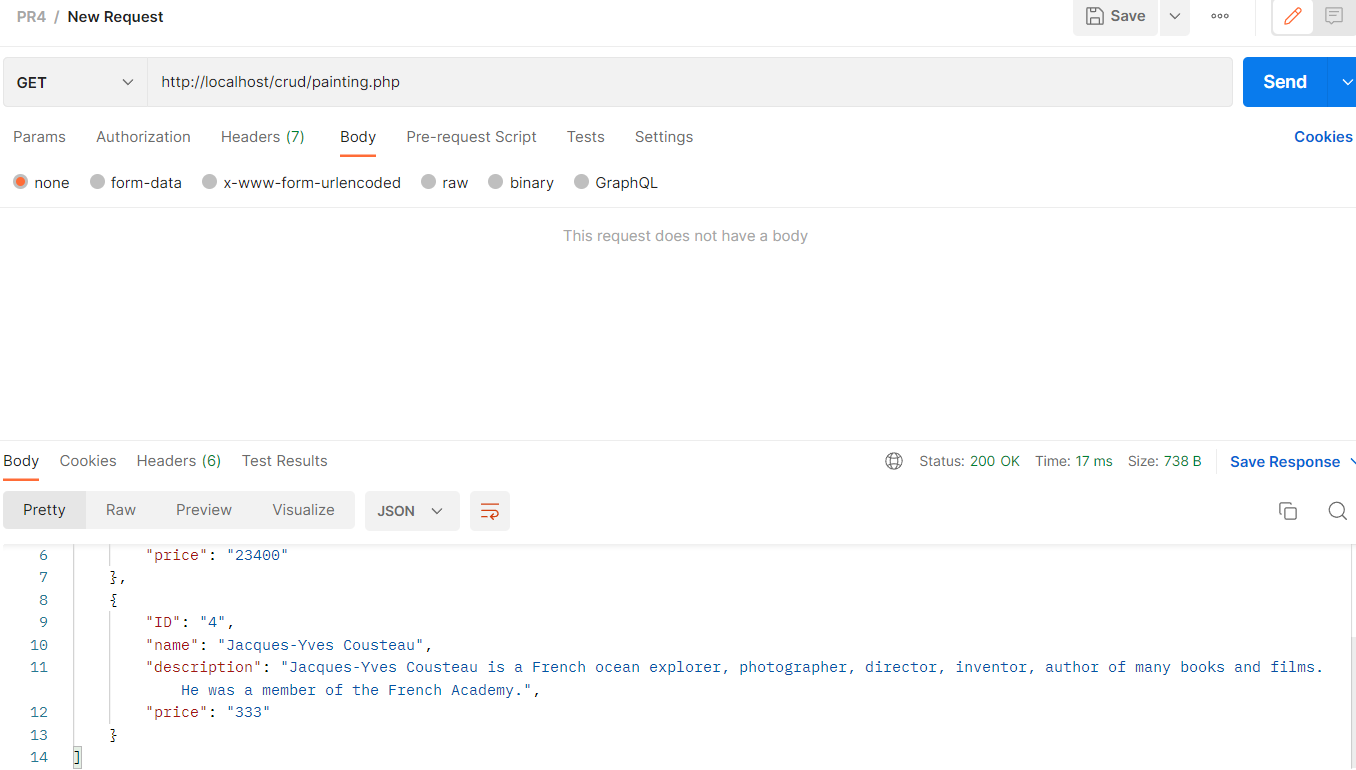


Рисунок 3 – GET-запрос

Далее проверим метод удаления записи (Рисунки 4-5).

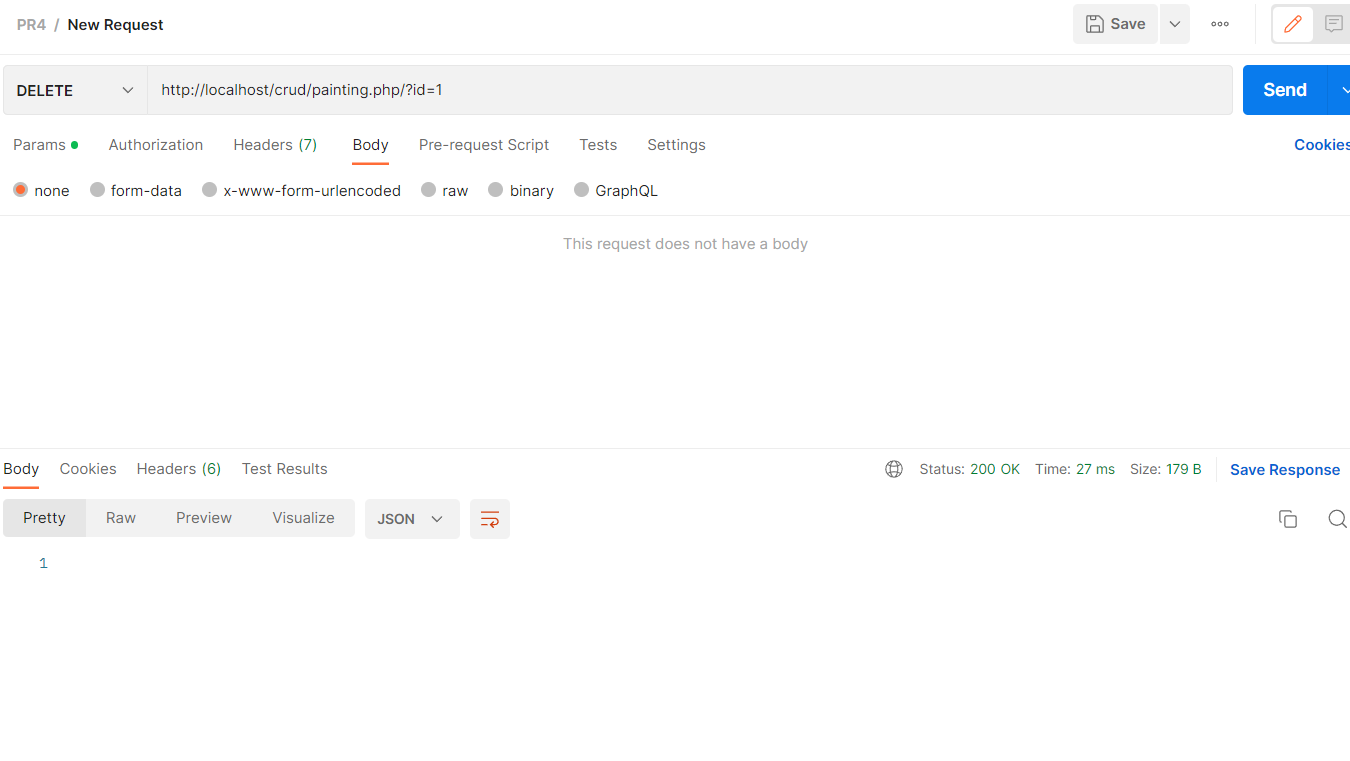


Рисунок 4 – DELETE-запрос

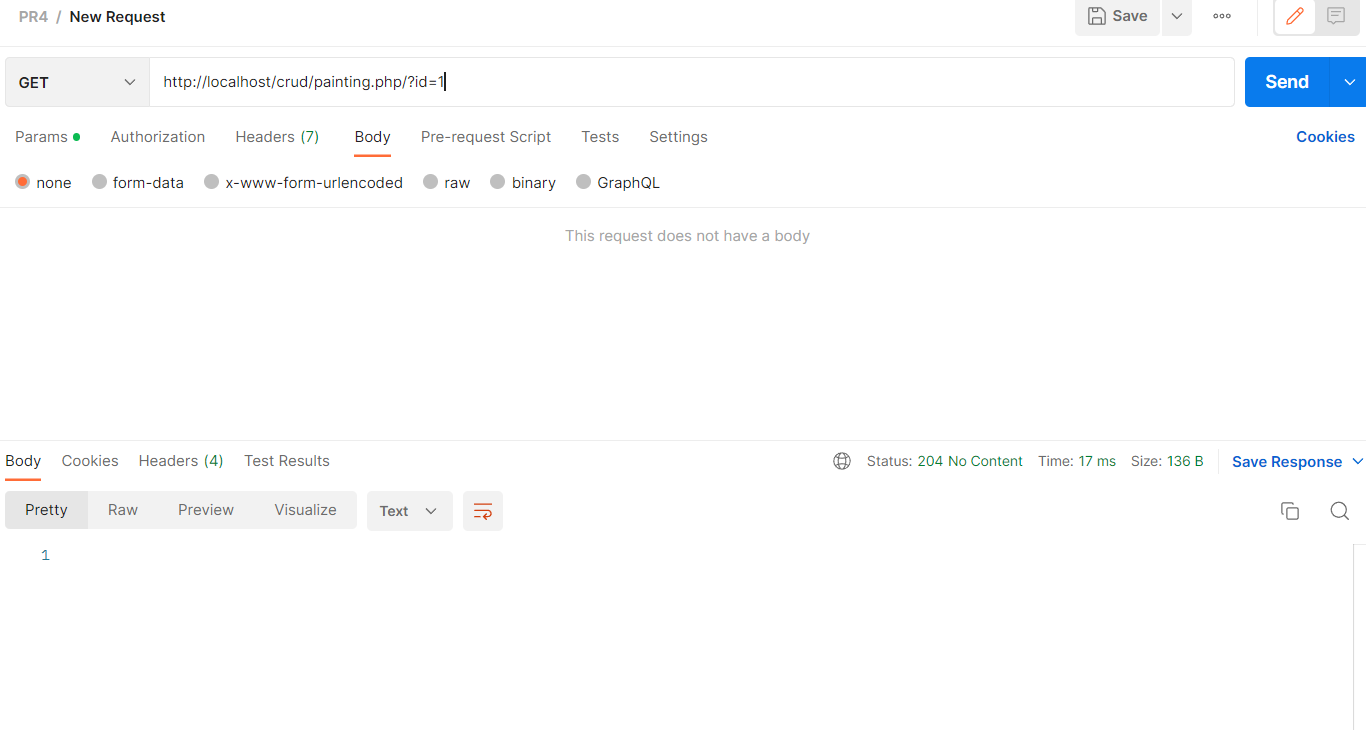


Рисунок 5 – GET-запрос

Теперь удостоверимся в работоспособности user.php (рисунок 6).

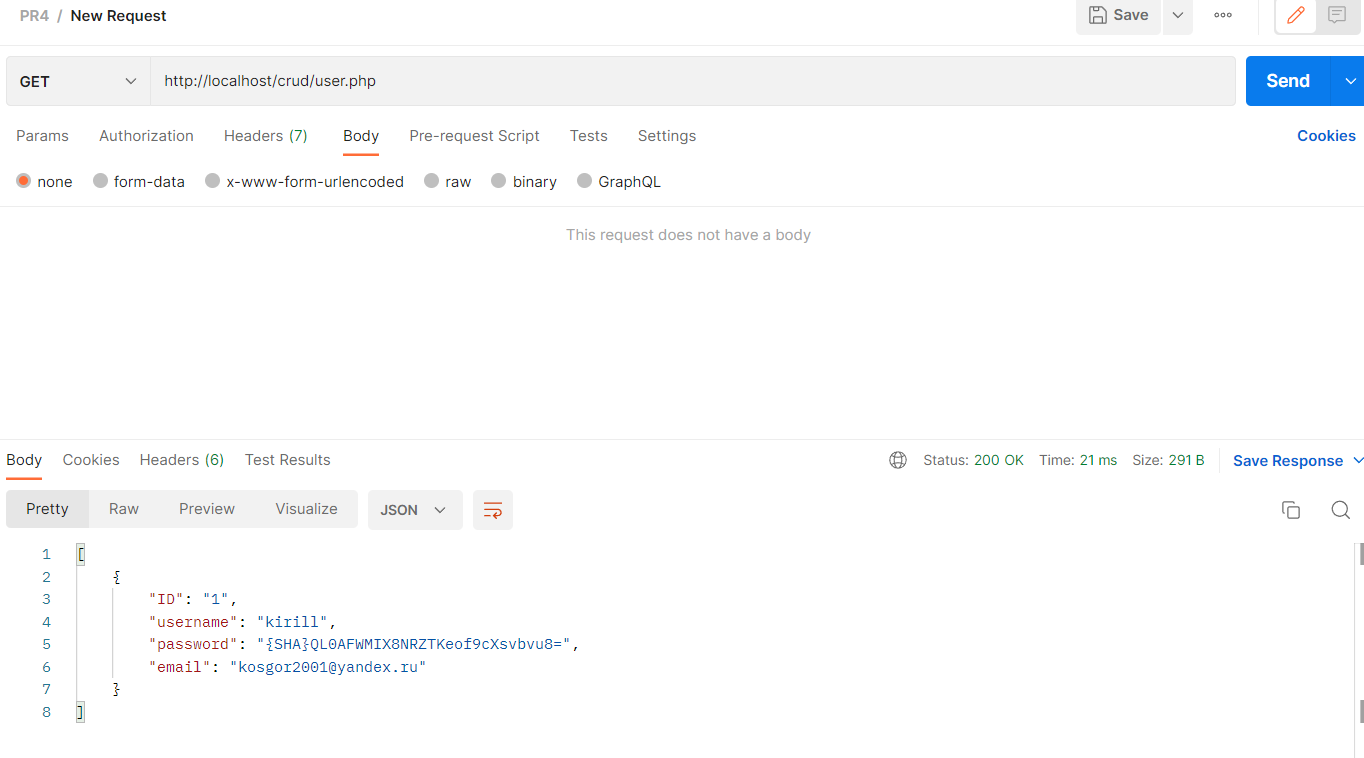


Рисунок 6 – GET-запрос

Произведем PUT-запрос (Рисунки 7-8).

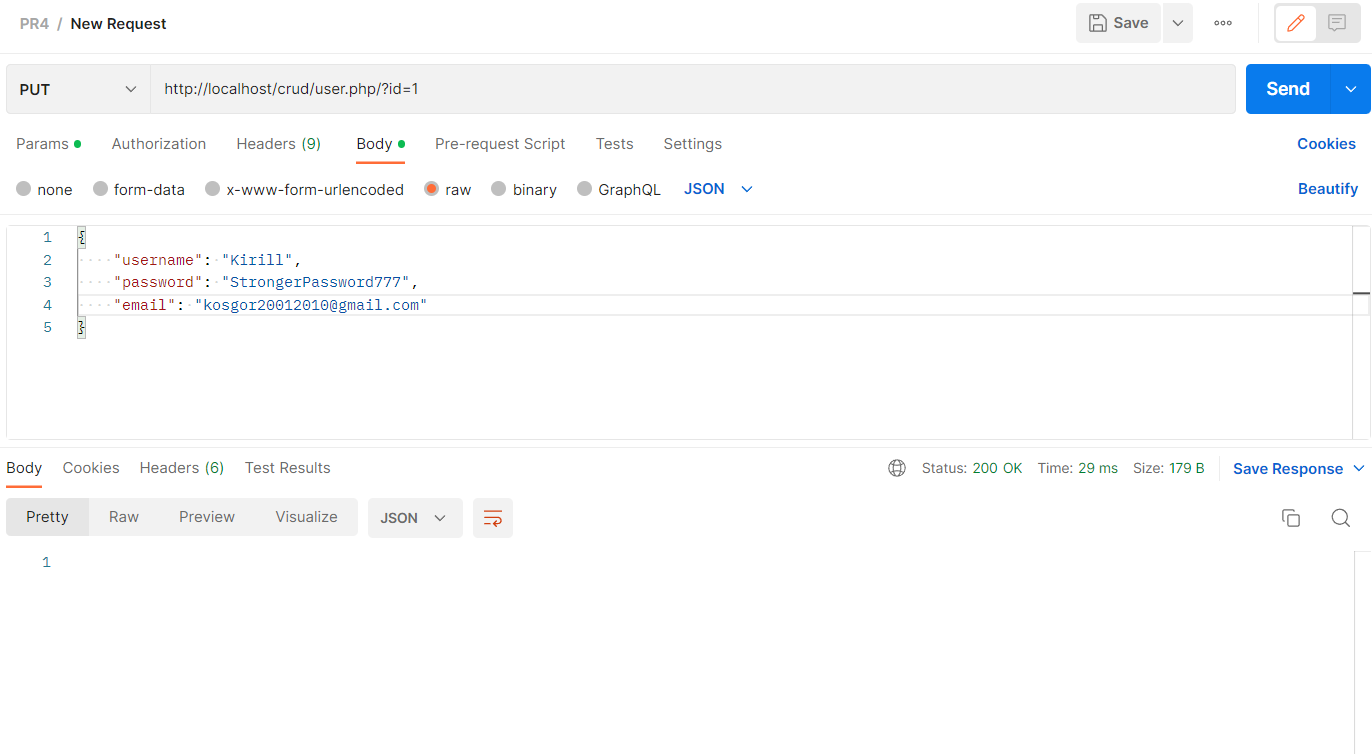


Рисунок 7 – PUT-запрос

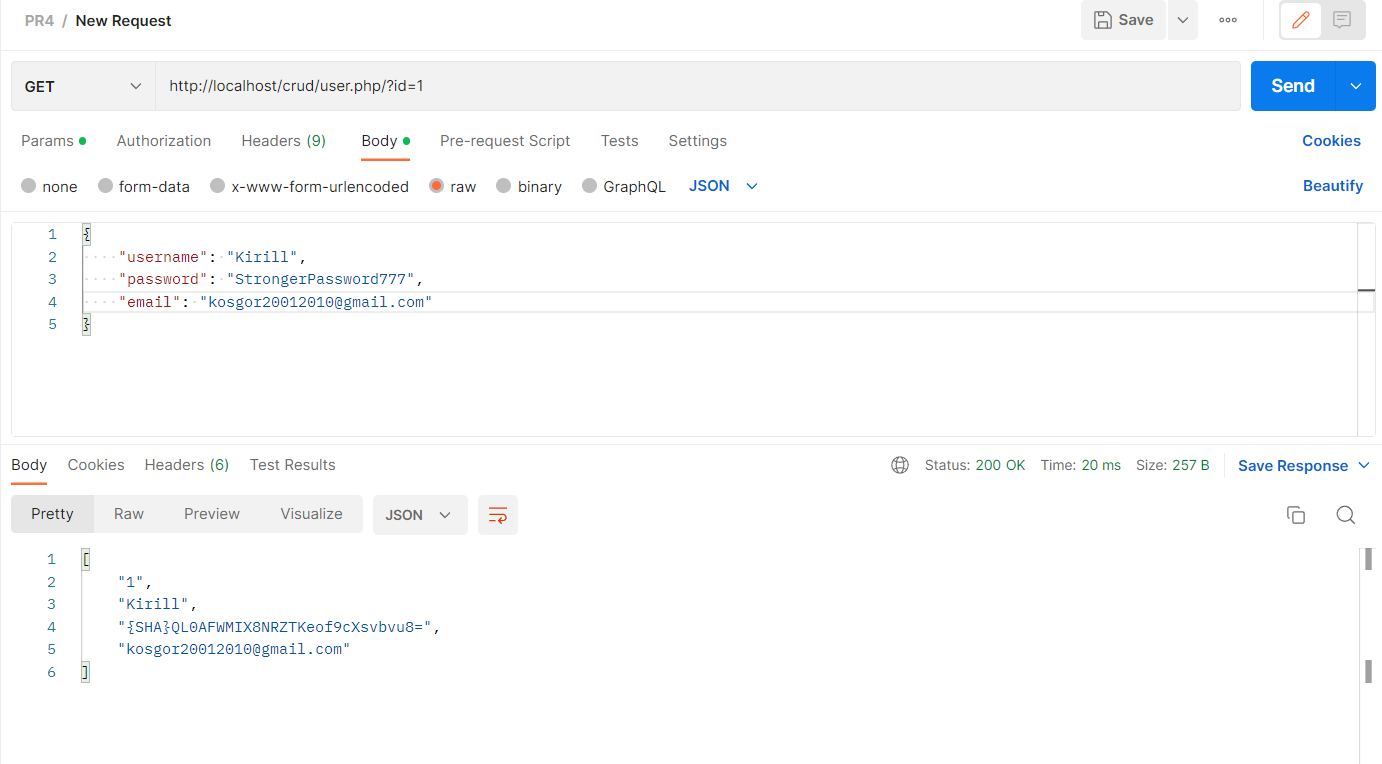


Рисунок 8 – GET-запрос

1. Вывод

В результате выполнения работы была создана сложная серверная конфигурация, состоящая из связки apache+nginx+php+MySQL, а также CRUD REST API. Получены практические навыки по настройке серверов apache и nginx.

1. Ответы на вопросы к практической работе

**1. Что такое HTTP-запрос?**

HTTP запросы - это сообщения, отправляемые клиентом, чтобы инициировать реакцию со стороны сервера.

**2. Опишите существующие HTTP-запросы.**

GET — получение ресурса

POST — создание ресурса

PUT — обновление ресурса

DELETE — удаление ресурса

**3. Опишите обработку запроса на PHP. Что нужно использовать, как вычленить параметры запроса?**

Внутри PHP-скрипта имеется несколько способов получения доступа к данным, переданным клиентом по протоколу HTTP. До версии PHP 4.1.0 доступ к таким данным осуществлялся по именам переданных переменных (напомним, что данные передаются в виде пар "имя переменной, символ "=", значение переменной"). Таким образом, если, например, было передано first\_name=Nina, то внутри скрипта появлялась переменная $first\_name со значением Nina. Если требовалось различать, каким методом были переданы данные, то использовались ассоциативные массивы $HTTP\_POST\_VARS и$HTTP\_GET\_VARS , ключами которых являлись имена переданных переменных, а значениями – соответственно значения этих переменных.

**4.Опишите создание HTML-форм на PHP.**

Поэтому требуется в первую очередь научиться получать данные из формы в сценарии .

В PHP это делается легко — все данные из формы находятся в глобальном ассоциативном массиве $\_POST.

Этот массив всегда будет неявно присутствовать в сценарии, если он был загружен по методу POST.

Каждое поле из формы будет находиться в массиве, где ключом будет значение атрибута name, а значением содержимое поля.

**5.Что такое API?**

API (Application Programming Interface или интерфейс программирования приложений) — это совокупность инструментов и функций в виде интерфейса для создания новых приложений, благодаря которому одна программа будет взаимодействовать с другой.

**6.Опишите API как средство интеграции приложений.**

Программные компоненты взаимодействуют друг с другом посредством API. При этом обычно компоненты образуют иерархию - высокоуровневые компоненты используют API низкоуровневых компонентов, а те, в свою очередь, используют API ещё более низкоуровневых компонентов.

**7.Что такое Web API?**

Веб-API - это интерфейс прикладного программирования для веб-сервера или веб-браузера. Это концепция веб-разработки, обычно ограниченная клиентской стороной веб-приложения (включая любые используемые веб-фреймворки), и поэтому обычно не включает детали реализации веб-сервера или браузера, такие как SAPI или API, если они не доступны для общего доступа через удаленное веб-приложение.

**8.Приведите пример API.**

APILayer,GoogleAPI,TelegramAPI,OnenWeatherAPI

**9. Что такое REST?**

REST — архитектурный стиль взаимодействия компонентов распределённого приложения в сети. Другими словами, REST — это набор правил того, как программисту организовать написание кода серверного приложения, чтобы все системы легко обменивались данными и приложение можно было масштабировать.

**10. Как организована передача данных в архитектуре REST?**

Отсутствие дополнительных внутренних прослоек означает передачу данных в том же виде, что и сами данные. Т.е. мы не заворачиваем данные в XML, как это делает SOAP и XML-RPC, не используем AMF, как это делает Flash и т.д. Просто отдаем сами данные.

**11. Как организована работа REST?**

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат

**12. Что такое SOAP?**

SOAP (от англ. Simple Object Access Protocol — простой протокол доступа к объектам) — протокол обмена структурированными сообщениями в распределённой вычислительной среде. Первоначально SOAP предназначался в основном для реализации удалённого вызова процедур (RPC). Сейчас протокол используется для обмена произвольными сообщениями в формате XML, а не только для вызова процедур.

**13. Чем SOAP отличается от REST?**

Отсутствие дополнительных внутренних прослоек означает передачу данных в том же виде, что и сами данные. Т.е. мы не заворачиваем данные в XML, как это делает SOAP и XML-RPC, не используем AMF, как это делает Flash и т.д. Просто отдаем сами данные.

**14. Для чего нужен SOAP-процессор?**

Для правильной обработки XML-сообщения процесс-«слушатель» HTTP (напр. Apache или Microsoft IIS) должен предоставить SOAP-процессор, или, другими словами, должен иметь возможность обрабатывать XML.

**15.Опишите общую структуру SOAP-сообщения.**

Сообщение SOAP выглядит так:

Envelope ,Header ,Body ,Fault

**16.Что такое и что содержит Конверт (SOAP Envelope)?**

Envelope — корневой элемент, который определяет сообщение и пространство имен, использованное в документе.

**17. Что такое и что содержит Заголовок SOAP (SOAP Header)?**

Header — содержит атрибуты сообщения, например: информация о безопасности или о сетевой маршрутизации.

**18. Что такое и что содержит Тело SOAP (SOAP Body)?**

Body — содержит сообщение, которым обмениваются приложения.

**19. Опишите SOAP-сообщение с вложением.**

Возможно, потребуется передать сообщение SOAP вместе с вложениями различного рода, начиная от факсимильных изображений юридических документов и заканчивая инженерными чертежами. Такие данные часто представлены в некотором двоичном формате. Например, большинство изображений в Интернете передаются с использованием форматов данных GIF или JPEG. В этом документе мы описываем стандартный способ связать SOAP-сообщение с одним или несколькими вложениями в их собственном формате в многокомпонентной структуре MIME для транспорта.

**20. Что такое graphql?**

GraphQL — язык запросов данных и язык манипулирования данными с открытым исходным кодом для построения веб ориентированных программных интерфейсов.

**21. Что такое Распознаватели (resolvers) в graphql?**

Resolver - это набор функций, которые генерируют ответ на запрос GraphQL.

**22. Из чего состоит экосистема graphql, что нужно, чтобы использовать данную технологию?**

В двух словах, GraphQL это синтаксис, который описывает как запрашивать данные, и, в основном, используется клиентом для загрузки данных с сервера.

**23. Что такое валидация данных и для чего она нужна?**

Валидация данных (англ. Data validation) — это процесс проверки данных различных типов по критериям корректности и полезности для конкретного применения.

Позволяет клиенту точно указать, какие данные ему нужны.

Облегчает агрегацию данных из нескольких источников.

Использует систему типов для описания данных.

**24. Где и когда выполнять валидацию данных?**

Но если всё-таки проверка нужна, логика подсказывает, что удобно проверять данные в том месте, где они попадают в программу из внешнего мира. После такой проверки можно быть уверенным, что в программу попадают правильные данные и в дальнейшем они могут использоваться без дополнительных проверок.Это может быть пользовательский интерфейс, через который человек вводит данные. Это может быть файл, содержащий настройки программы или данные, которые программа должна обработать. Это может быть база данных, в которую информация может попадать из других программ. Это может быть сетевой протокол обмена данными с другими программами. Наконец, это может быть программный интерфейс, который использует другая программа, вызывая некоторые функции/процедуры и передавая в них параметры.

**25. Как выполнять валидацию данных?**

Посимвольная проверка. Как правило такие проверки выполняются в пользовательском интерфейсе, по мере ввода данных. Но не только. Например, лексический анализатор компилятора тоже выявляет недопустимые символы непосредственно в процессе чтения компилируемого файла. Поэтому такие проверки можно условно назвать «лексическими».

**26. Приведите пример с поэтапной валидацией данных.**

Регистрация

**27. Что такое запрос и мутация в graphql и чем они отличаются?**

Запросы GraphQL — это сущности, представляющие собой запрос к серверу на получение неких данных. Например, у нас есть некий пользовательский интерфейс, который мы хотим заполнить данными. За этими данными мы и обращаемся к серверу, выполняя запрос. При использовании традиционных REST API наш запрос принимает вид GET-запроса.

В то время как запросы GraphQL выполняют загрузку данных, мутации ответственны за внесение в данные изменений. Мутации могут быть использованы в виде базового механизма RPC (Remote Procedure Call, вызов удалённых процедур) для решения различных задач наподобие отправки данных пользователя API стороннего разработчика

1. Список использованной литературы
2. Официальная документация докера [Электронный ресурс] – URL: <https://docs.docker.com> (дата обращения 03.09.2022)
3. Установка и настройка PHP [Электронный ресурс] – URL: <https://www.php.net/manual/ru/install.php> (дата обращения 03.09.2022)
4. Docker Compose [Электронный ресурс] – URL: <https://habr.com/ru/company/ruvds/blog/450312> (дата обращения 03.09.2022)
5. Docker Hub [Электронный ресурс] – URL: <https://hub.docker.com> (дата обращения 03.09.2022)