|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 4** | |
| **по дисциплине** | |
| «Разработка серверных частей интернет-ресурсов»  **Тема практической работы: «**Реализация взаимодействия клиента и сервера с использованием технологии API» | |
|  | |
| Выполнил студент группы ИКБО-24-20 | Краев И. А. |
| Руководитель практической работы | Благирев М.М. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2022

Оглавление

[Цель работы 3](#_Toc117689831)

[Постановка задачи 3](#_Toc117689832)

[Ход работы 3](#_Toc117689833)

[Ответы на вопросы к практической работе 12](#_Toc117689834)

[Список использованной литературы 18](#_Toc117689835)

# 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 №3. То есть интерфейс предлагается создать уже в существующем веб-приложении. Также предполагается использование темы практической̆ работы №3 для продолжения модернизирования собственной̆ системы. Изменение темы согласовывается отдельно с преподавателем. Хранение данных предполагается уже в существующей̆ базе данных.

# 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.

# Ход работы

К данной практической работы мной была создана структура проекта и контейнеров на рисунках 1-2.

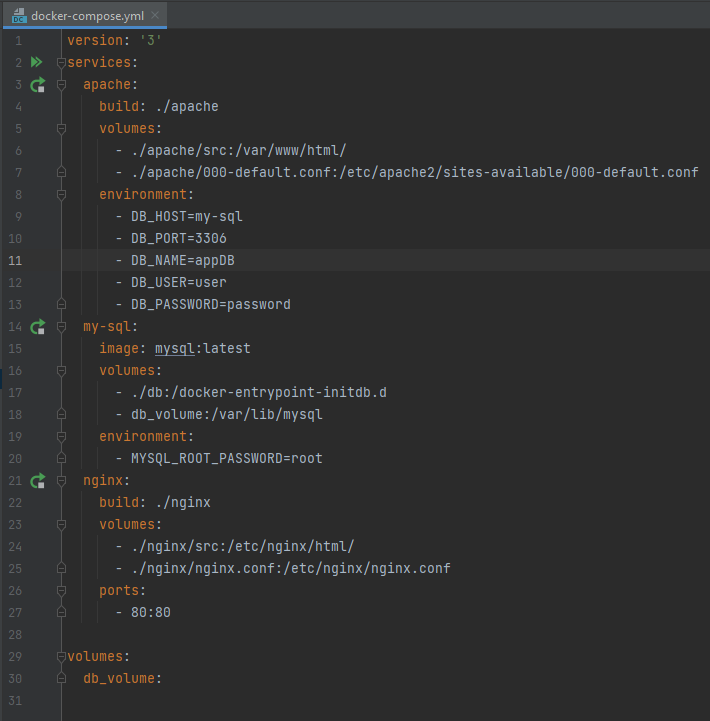


Рисунок 1 – Структура контейнеров

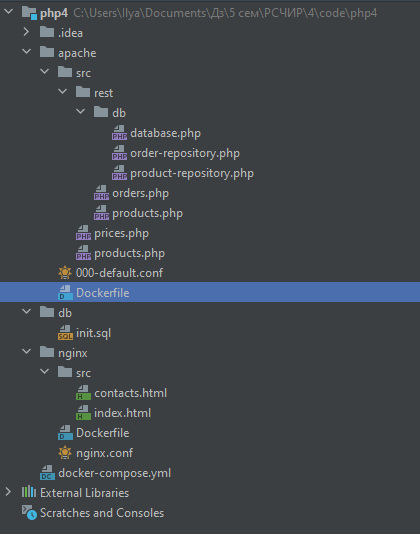


Рисунок 2 – Структура проекта

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 рисунке 3.

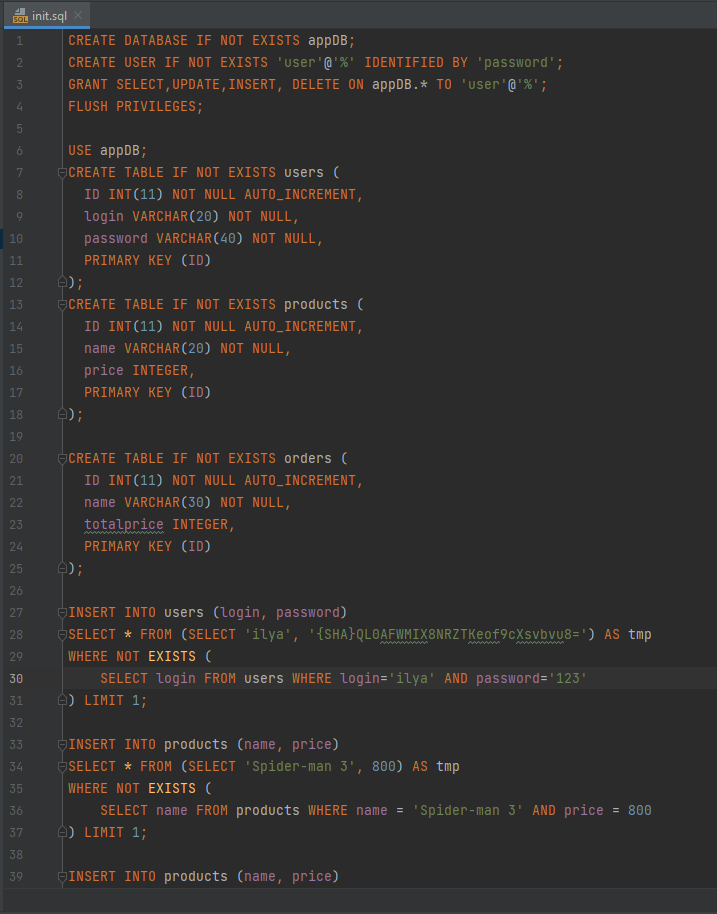


Рисунок 3 – Файл init.sql

Далее запускается образ php-apache на основе 3 практики. Был написан REST API для сущностей users и tickets (Листинг 1-4).

Листинг 1 – файл products.php

<?php

include('db/product-repository.php');

// необходимые HTTP-заголовки

header("Content-Type: application/json; charset=UTF-8");

$method = $\_SERVER['REQUEST\_METHOD'];

switch ($method) {

case 'GET':

echo json\_encode(ProductRepository::read());

break;

case 'POST':

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

ProductRepository::create($newProduct->name, $newProduct->price);

break;

case 'PUT':

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

echo ProductRepository::update($newProduct->id, $newProduct->name, $newProduct->price);

break;

case 'DELETE':

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

echo ProductRepository::delete($oldProduct->id);

break;

}

?>

Листинг 2 –файл product-repository.php

<?php

include('database.php');

class Product{

public $id;

public $name;

public $price;

function \_\_construct($p\_name, $p\_price, $p\_id=0) {

$this->id =$p\_id;

$this->name = $p\_name;

$this->price = $p\_price;

}

}

class ProductRepository{

public static function create($name, $price) {

$mysqli = ConnectionDB::getInstance();

return $mysqli->query("INSERT INTO products (name, price) VALUES ('$name', '$price')");

}

public static function read() {

$response = [];

$mysqli = ConnectionDB::getInstance();

$result = $mysqli->query("SELECT \* FROM products");

foreach ($result as $row){

$response[count($response)] = new Product($row['name'], $row['price'], $row['ID']);

}

return $response;

}

public static function update($id, $name, $price) {

$mysqli = ConnectionDB::getInstance();

return $mysqli->query("UPDATE products SET name = '$name', price = '$price' WHERE id = '$id'");

}

Продолжение листинга 2.

public static function delete($id) {

$mysqli = ConnectionDB::getInstance();

return $mysqli->query("DELETE FROM products where id = '$id'");

}

}

?>

Листинг 3 – файл orders.php

<?php

include('db/order-repository.php');

// необходимые HTTP-заголовки

header("Content-Type: application/json; charset=UTF-8");

$method = $\_SERVER['REQUEST\_METHOD'];

switch ($method) {

case 'GET':

echo json\_encode(OrderRepository::read());

break;

case 'POST':

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

OrderRepository::create($newOrder->name, $newOrder->totalprice);

break;

case 'PUT':

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

echo OrderRepository::update($newOrder->id, $newOrder->name, $newOrder->totalprice);

break;

case 'DELETE':

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

echo OrderRepository::delete($oldOrder->id);

break;

}

?>

Листинг 4 – файл order-repository.php

<?php

include('database.php');

class Order{

public $id;

public $name;

public $totalprice;

function \_\_construct($d\_name, $d\_totalprice,$d\_id=0) {

$this->id =$d\_id;

$this->name = $d\_name;

$this->totalprice = $d\_totalprice;

}

}

class OrderRepository{

public static function create($name, $totalprice) {

$mysqli = ConnectionDB::getInstance();

return $mysqli->query("INSERT INTO orders (name, totalprice) VALUES ('$name', '$totalprice')");

}

public static function read() {

$response = [];

$mysqli = ConnectionDB::getInstance();

$result = $mysqli->query("SELECT \* FROM orders");

foreach ($result as $row){

$response[count($response)] = new Order($row['name'], $row['totalprice'], $row['ID']);

}

return $response;

}

public static function update($id, $name, $totalprice) {

$mysqli = ConnectionDB::getInstance();

return $mysqli->query("UPDATE orders SET name = '$name', totalprice = '$totalprice' WHERE id = '$id'");

}

Продолжение листинга 4.

public static function delete($id) {

$mysqli = ConnectionDB::getInstance();

return $mysqli->query("DELETE FROM orders where id = '$id'");

}

}

?>

Теперь запустим сервер и проверим работоспособность сайта (Рисунки 5-8).

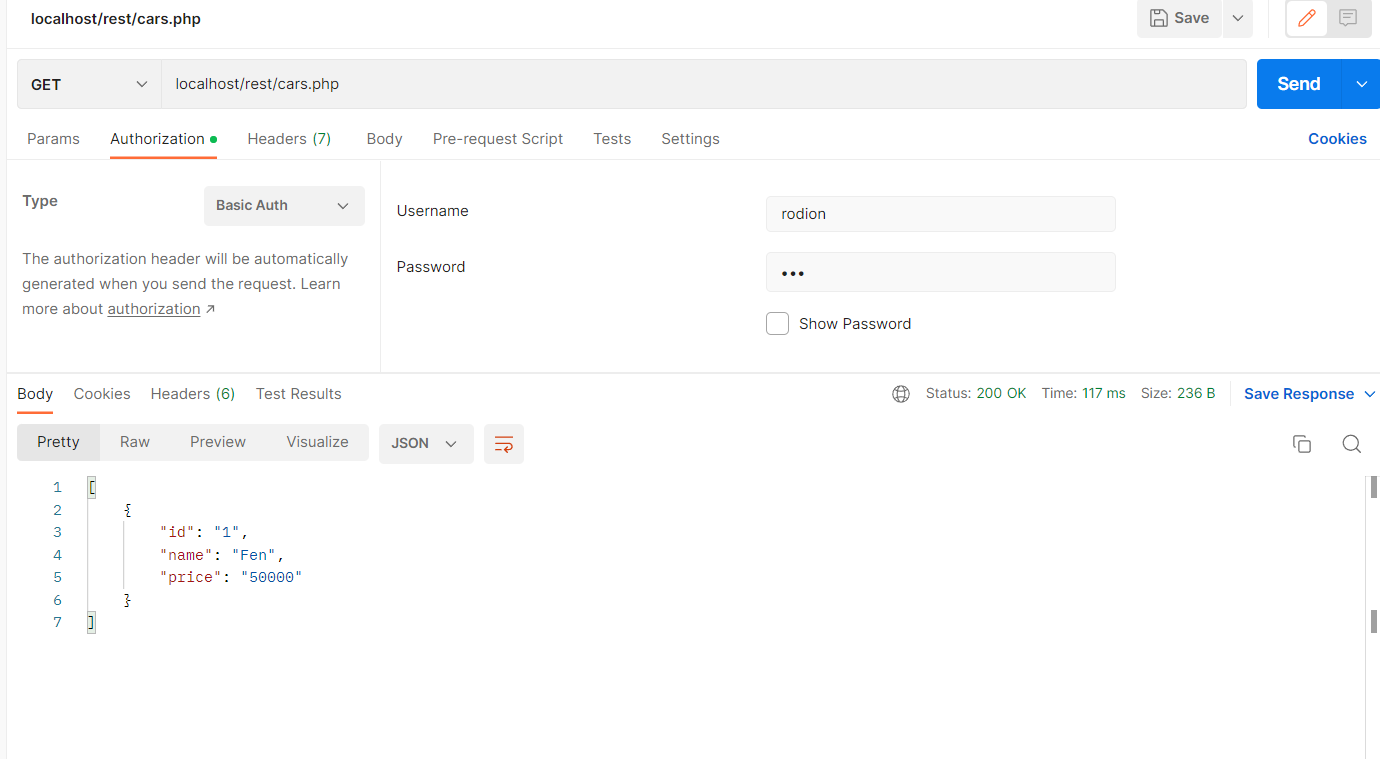


Рисунок 8 – GET-запрос

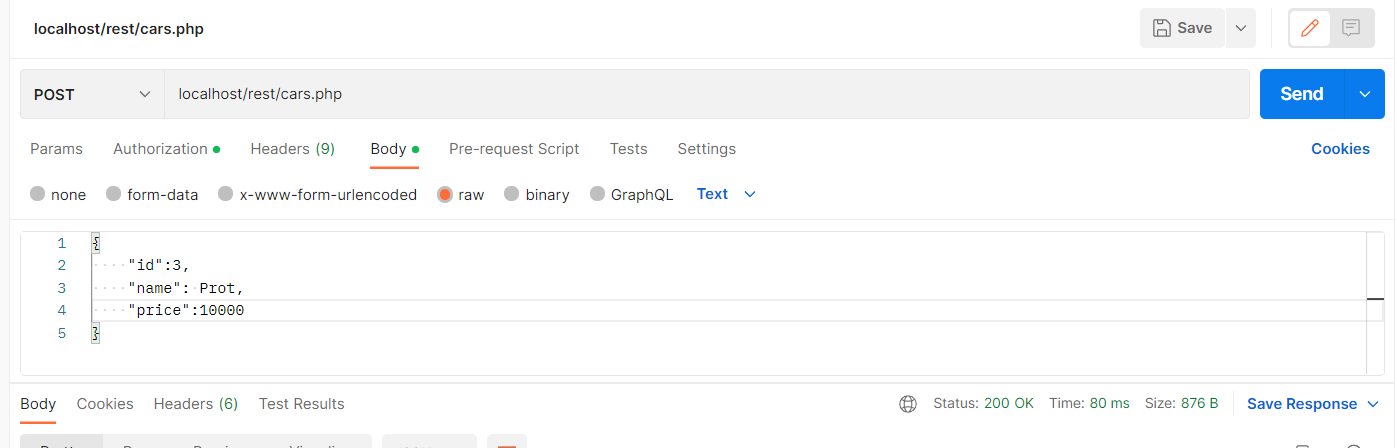


Рисунок 9 – POST-запрос

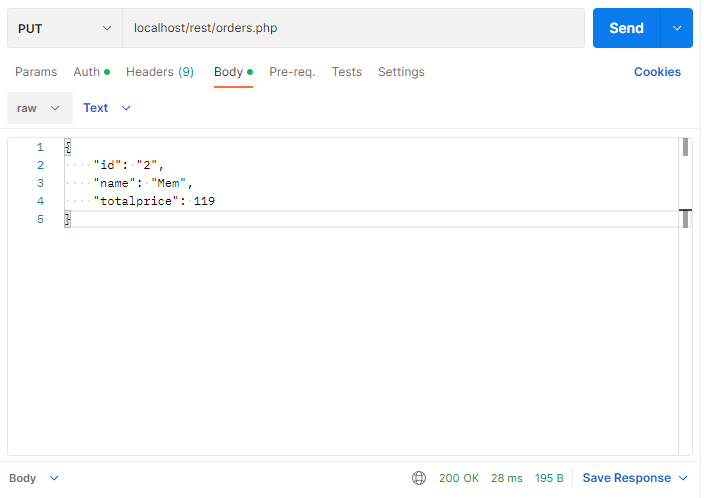


Рисунок 10 – PUT-запрос

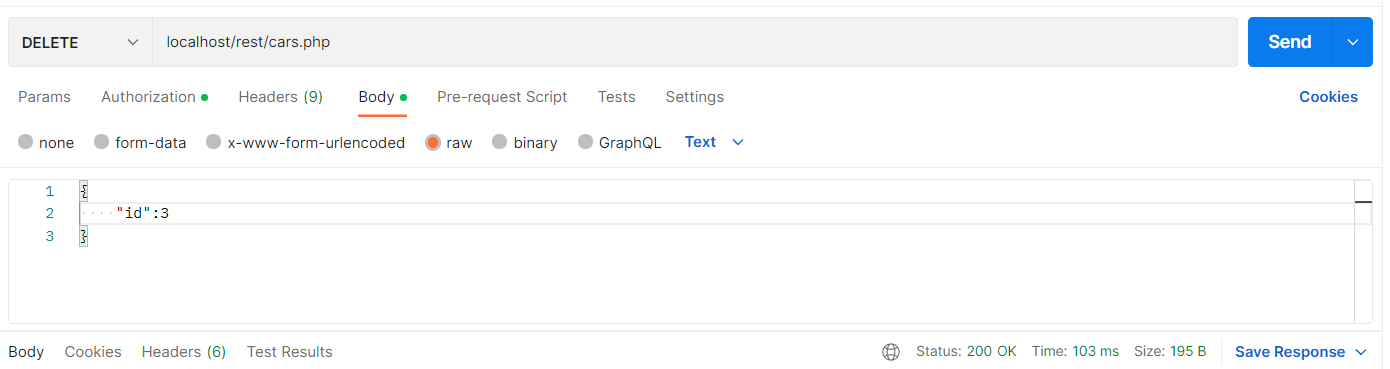


Рисунок 11 – DELETE-запрос

**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.

# 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?**

На каждый адрес задаётся обработчик (функция, handler), который выполняет необходимые действия и возвращает ответ. Этот объект содержит методы на каждый глагол HTTP: get, post, put и так далее. Эти методы принимают на вход два параметра, первый — адрес, для которого вызовется обработчик и второй, собственно, сам обработчик. Лямбда-функция с двумя параметрами $request и $response.

$HTTP\_GET\_VARS, ключами которых являлись имена переданных переменных, а значениями – соответственно значения этих переменных.

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

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 – это механизмы, которые позволяют двум программным компонентам взаимодействовать друг с другом, используя набор определений и протоколов.

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

Если программу (модуль, библиотеку) рассматривать как чёрный ящик, то API — **это набор «ручек», которые доступны пользователю данного ящика и которые он может вертеть и дёргать**. Программные компоненты взаимодействуют друг с другом посредством API.

**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, 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 — это протокол, по которому веб-сервисы взаимодействуют друг с другом или с клиентами. При этом сообщения должны быть написаны на языке [XML](https://blog.skillfactory.ru/glossary/xml/) в соответствии со строгими стандартами, иначе сервер вернет ошибку.

**13. Чем SOAP отличается от REST?**

REST и SOAP на самом деле не сопоставимы. REST — это архитектурный стиль. SOAP — это формат обмена сообщениями.

* Пример реализации **RESTful**: JSON через HTTP
* Пример реализации **SOAP**: XML поверх SOAP через HTTP

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

Для правильной обработки XML-сообщения процесс-«слушатель» HTTP 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 — 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?**

Если 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.

**25. Как выполнять валидацию данных?**

**Посимвольная проверка.** Как правило такие проверки выполняются в пользовательском интерфейсе, по мере ввода данных.

**Проверка отдельных значений.** Для пользовательского интерфейса это проверка значения в отдельном поле, причём выполняться она может как по мере ввода, так и после завершения ввода, когда поле теряет фокус.

**Совокупность входных значений.** Можно предположить, что в программу сначала передаются какие-то данные, после чего подаётся некоторый сигнал, который инициирует их обработку.

**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. 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 вызова удалённых процедур для решения различных задач наподобие отправки данных пользователя API стороннего разработчика.

# Ссылка на удаленный репозиторий проекта:

https://github.com/AIdaveloper/RSCHIR\_3.1/tree/main/4

# Список использованной литературы

1. Официальная документация докера [Электронный ресурс] – URL: <https://docs.docker.com> (дата обращения 03.11.2022)
2. Установка и настройка PHP [Электронный ресурс] – URL: <https://www.php.net/manual/ru/install.php> (дата обращения 03.11.2022)
3. Docker Compose [Электронный ресурс] – URL: <https://habr.com/ru/company/ruvds/blog/450312> (дата обращения 03.11.2022)
4. Docker Hub [Электронный ресурс] – URL: <https://hub.docker.com> (дата обращения 03.11.2022)