|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА** – **Российский технологический университет»**  **РТУ МИРЭА** | | | | |
| Институт информационных технологий | |
| Кафедра Инструментального и прикладного программного обеспечения (ИиППО) | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ПРАКТИЧЕСКАЯ РАБОТА №4** | | | |
| по дисциплине «Разработка серверных частей интернет-ресурсов» | | | |
| **Настройка сервера по заданным характеристикам, размещение проекта** | | | |
| **Студент группы ИКБО-16-19** | | Мурадов Н.Н.  ­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) | |
|  | |  | |
| Руководитель практической работы | | Преподаватель Волков М.Ю.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |
| Работа представлена | «\_\_\_»\_\_\_\_\_\_\_2021 г. | | (подпись руководителя) |
| Допущен к работе | «\_\_\_»\_\_\_\_\_\_\_2021 г. | |  |
|  |  | |  |

Москва 2021

Ход работы

[Задание 3](#_Toc86844531)

[Упражнение 3](#_Toc86844532)

[Вывод: 11](#_Toc86844533)

[Вопросы: 11](#_Toc86844534)

[Список использованной литературы: 20](#_Toc86844535)

## Задание

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 практической работы №3 для продолжения модернизирования собственной системы. Изменение темы согласовывается отдельно с преподавателем. Хранение данных предполагается уже в существующей базе данных.

## Упражнение

Спецификация:

1. метод GET localhost:4545/graphql.php - получить данные из бд
2. метод POST localhost:4545/graphql.php - добавить данные в бд
3. метод DELETE localhost:4545/graphql.php - удалить данные из бд
4. метод PATCH localhost:4545/graphql.php - обновить данные в бд

**docker-compose.yml:**

version: "3"

services:

apache:

build:

context: httpd

ports:

- 4545:80

volumes:

- ./src:/var/www/html

- ./log:/var/log/apache2

depends\_on:

- bd

nginx:

image: nginx:1.13

ports:

- 4542:80

volumes:

- ./src:/var/www/html

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

depends\_on:

- apache

bd:

image: mysql:5.7

volumes:

- ./bd/data:/var/lib/mysql

- ./bd:/docker-entrypoint-initdb.d

ports:

- 3306:3306

environment:

MYSQL\_ROOT\_PASSWORD: qwerty

**httpd/Dockerfile:**

FROM composer:latest AS composer

FROM php:7.2-apache

RUN apt-get update && apt-get install -y zlib1g-dev && apt-get install -y libzip-dev && apt-get install -y unzip

RUN docker-php-ext-install pdo pdo\_mysql mysqli zip

COPY composer.json /var/www/html/composer.json

ENV COMPOSER\_ALLOW\_SUPERUSER 1

COPY --from=composer /usr/bin/composer /usr/bin/composer

RUN composer dumpautoload

**src/graphql.php:**

<?php

require\_once \_\_DIR\_\_ . '/vendor/autoload.php';

require\_once 'Bd/Types.php';

require\_once 'Bd/DB.php';

use Bd\Types;

use Bd\DB;

use GraphQL\GraphQL;

use GraphQL\Type\Definition\ObjectType;

use GraphQL\Type\Schema;

try {

$config = [

'host' => 'bd',

'database' => 'appDB',

'username' => 'root',

'password' => 'qwerty'

];

DB::init($config);

$mutationType = new ObjectType([

'name' => 'Mutation',

'fields' => [

'sum' => [

'type' => Types::int(),

'args' => [

'x' => ['type' => Types::int()],

'y' => ['type' => Types::int()],

],

'resolve' => static function ($calc, array $args): int {

return $args['x'] + $args['y'];

},

],

],

]);

$schema = new Schema([

'query' => Types::query(),

'mutation' => $mutationType,

]);

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

$input = json\_decode($rawInput, true);

$query = $input['query'];

$variableValues = $input['variables'] ?? null;

$rootValue = ['prefix' => 'You said: '];

$result = GraphQL::executeQuery($schema, $query, $rootValue, null, $variableValues);

$output = $result->toArray();

} catch (Throwable $e) {

$output = [

'error' => [

'message' => $e->getMessage(),

],

];

}

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

echo json\_encode($output);

**src/Bd/DB.php:**

<?php

namespace Bd;

use PDO;

class DB

{

private static $pdo;

public static function init($config)

{

self::$pdo = new PDO("mysql:host={$config['host']};dbname={$config['database']}", $config['username'], $config['password']);

self::$pdo->setAttribute(PDO::ATTR\_DEFAULT\_FETCH\_MODE, PDO::FETCH\_OBJ);

}

public static function selectOne($query)

{

$records = self::select($query);

return array\_shift($records);

}

public static function select($query)

{

$statement = self::$pdo->query($query);

return $statement->fetchAll();

}

public static function query\_user($query)

{

$statement = self::$pdo->query($query);

$statement = self::$pdo->query('SELECT \* from users');

return $statement->fetchAll();

}

public static function affectingStatement($query)

{

$statement = self::$pdo->query($query);

return $statement->rowCount();

}

}

**src/Bd/Types.php:**

<?php

namespace Bd;

require\_once 'Bd/Type/QueryType.php';

require\_once 'Bd/Type/UserType.php';

use Bd\Type\QueryType;

use Bd\Type\UserType;

use GraphQL\Type\Definition\Type;

class Types

{

private static $query;

public static function query()

{

return self::$query ?: (self::$query = new QueryType());

}

public static function string()

{

return Type::string();

}

public static function int()

{

return Type::int();

}

public static function listOf($type)

{

return Type::listOf($type);

}

private static $user;

public static function user()

{

return self::$user ?: (self::$user = new UserType());

}

}

**src/Bd/Type/UserType.php:**

<?php

namespace Bd\Type;

require\_once 'Bd/DB.php';

require\_once 'Bd/Types.php';

use Bd\DB;

use Bd\Types;

use GraphQL\Type\Definition\ObjectType;

class UserType extends ObjectType

{

public function \_\_construct()

{

$config = [

'description' => 'Пользователь',

'fields' => function() {

return [

'id' => [

'type' => Types::string(),

'description' => 'Идентификатор пользователя'

],

'name' => [

'type' => Types::string(),

'description' => 'Имя пользователя'

],

'email' => [

'type' => Types::string(),

'description' => 'E-mail пользователя'

]

];

}

];

parent::\_\_construct($config);

}

}

**src/Bd/Type/QueryType.php:**

<?php

namespace Bd\Type;

require\_once 'Bd/DB.php';

require\_once 'Bd/Types.php';

use Bd\DB;

use Bd\Types;

use GraphQL\Type\Definition\ObjectType;

class QueryType extends ObjectType

{

public function \_\_construct()

{

$config = [

'fields' => function() {

return [

'user' => [

'type' => Types::user(),

'description' => 'Возвращает пользователя по id',

'args' => [

'id' => Types::int()

],

'resolve' => function ($root, $args) {

if($\_SERVER['REQUEST\_METHOD'] == 'GET')

{

return DB::selectOne("SELECT \* from users WHERE id = {$args['id']}");

}

echo 'Only GET method!';

}

],

'allUsers' => [

'type' => Types::listOf(Types::user()),

'description' => 'Список пользователей',

'resolve' => function () {

if($\_SERVER['REQUEST\_METHOD'] == 'GET')

{

return DB::select('SELECT \* from users');

}

echo 'Only GET method!';

}

],

'Post' => [

'type' => Types::listOf(Types::user()),

'description' => 'Добавление пользователей',

'args' => [

'id' => ['type' => Types::int()],

'name' => ['type' => Types::string()],

'email' => ['type' => Types::string()]

],

'resolve' => function ($uzer, array $args) {

$id = $args['id'];

$name = $args['name'] . '';

$email = $args['email'] . '';

if($\_SERVER['REQUEST\_METHOD'] == 'POST')

{

return DB::query\_user("INSERT INTO users (id, name, email) VALUES ({$id}, '{$name}', '{$email}')");

}

echo 'Only POST method!';

}

],

'Delete' => [

'type' => Types::listOf(Types::user()),

'description' => 'Добавление пользователей',

'args' => [

'id' => ['type' => Types::int()]

],

'resolve' => function ($uzer, array $args) {

$id = $args['id'];

if($\_SERVER['REQUEST\_METHOD'] == 'DELETE')

{

return DB::query\_user("DELETE FROM users WHERE id = {$id}");

}

echo 'Only DELETE method!';

}

],

'Patch' => [

'type' => Types::listOf(Types::user()),

'description' => 'Добавление пользователей',

'args' => [

'old\_id' => ['type' => Types::int()],

'new\_id' => ['type' => Types::int()],

'name' => ['type' => Types::string()],

'email' => ['type' => Types::string()]

],

'resolve' => function ($uzer, array $args) {

$old\_id = $args['old\_id'];

$new\_id = $args['new\_id'];

$name = $args['name'] . '';

$email = $args['email'] . '';

if($\_SERVER['REQUEST\_METHOD'] == 'PATCH')

{

return DB::query\_user("UPDATE users SET id={$new\_id}, name='{$name}', email='{$email}' WHERE id={$old\_id};");

}

echo 'Only PATCH method!';

}

]

];

}

];

parent::\_\_construct($config);

}

}

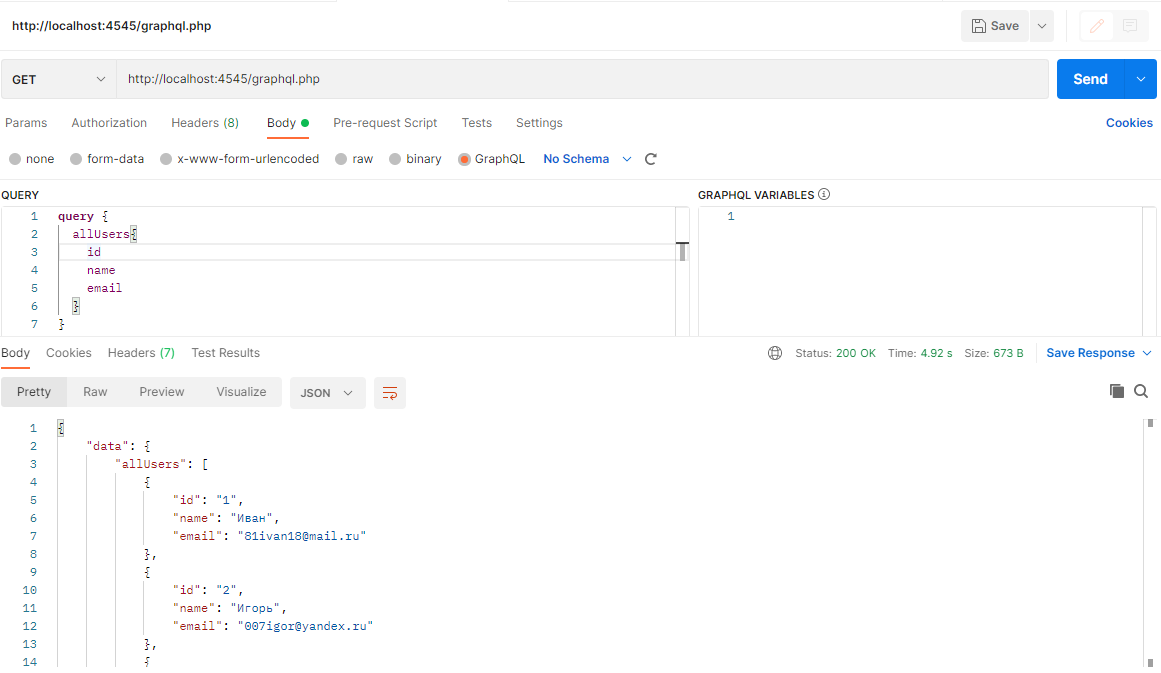


Рис.1-проверка запроса

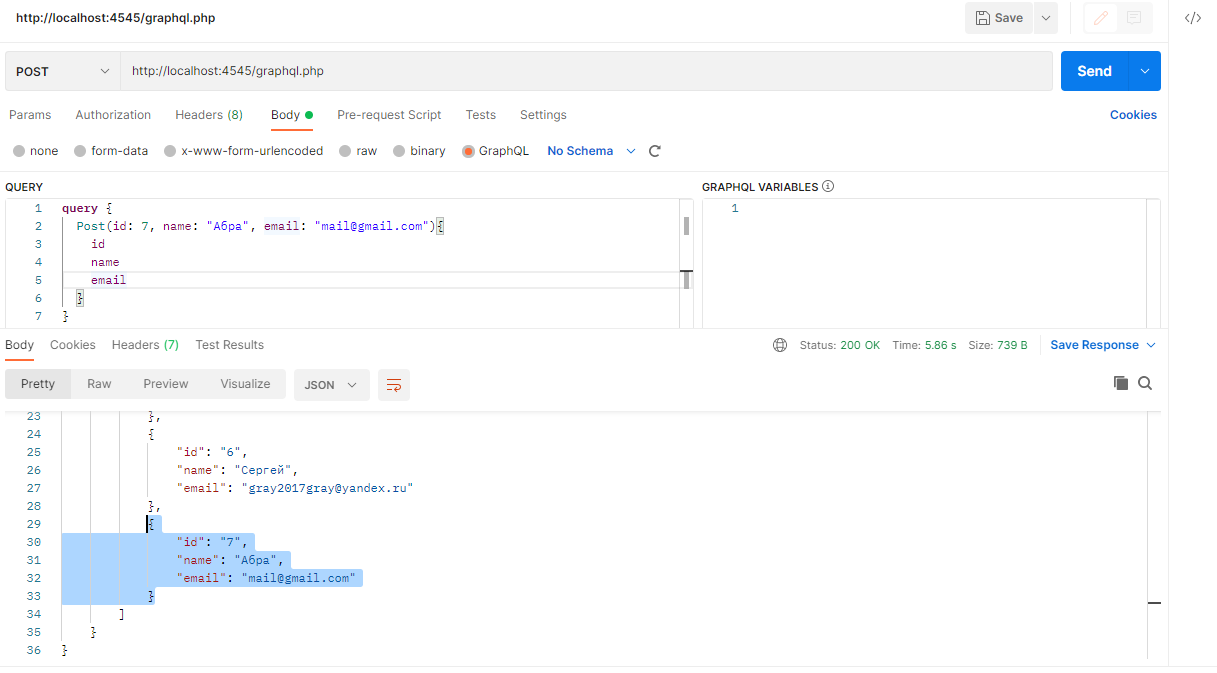


Рис.2-проверка запроса

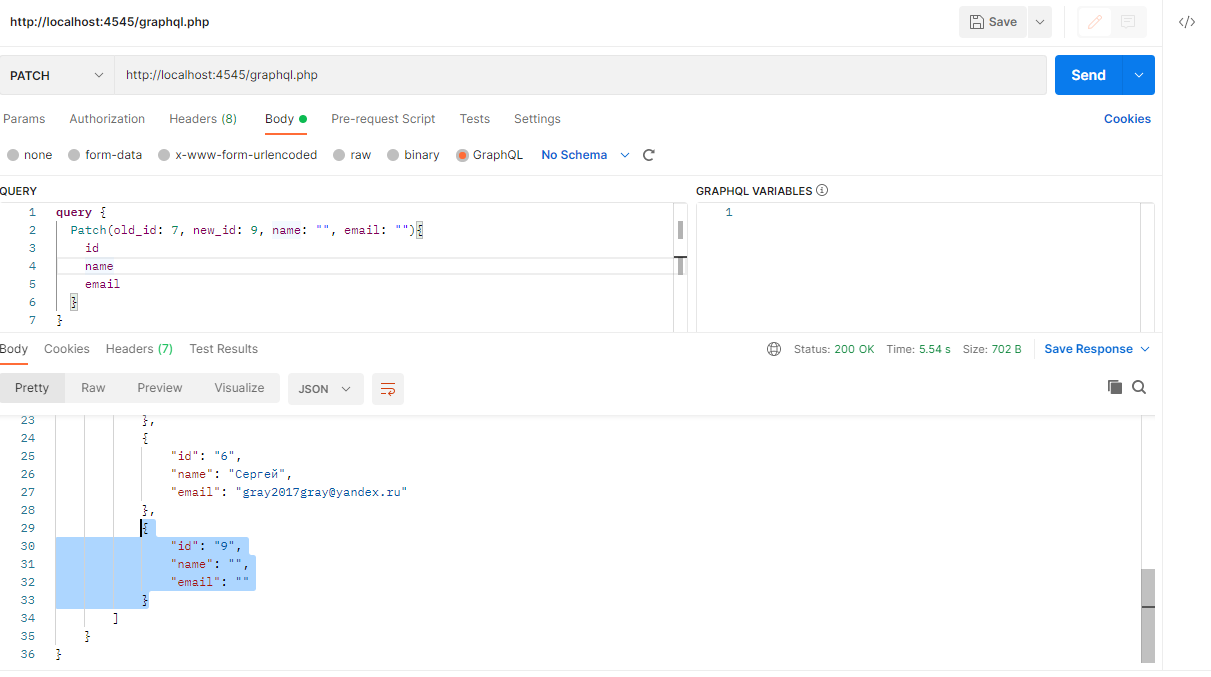


Рис.3-проверка запроса

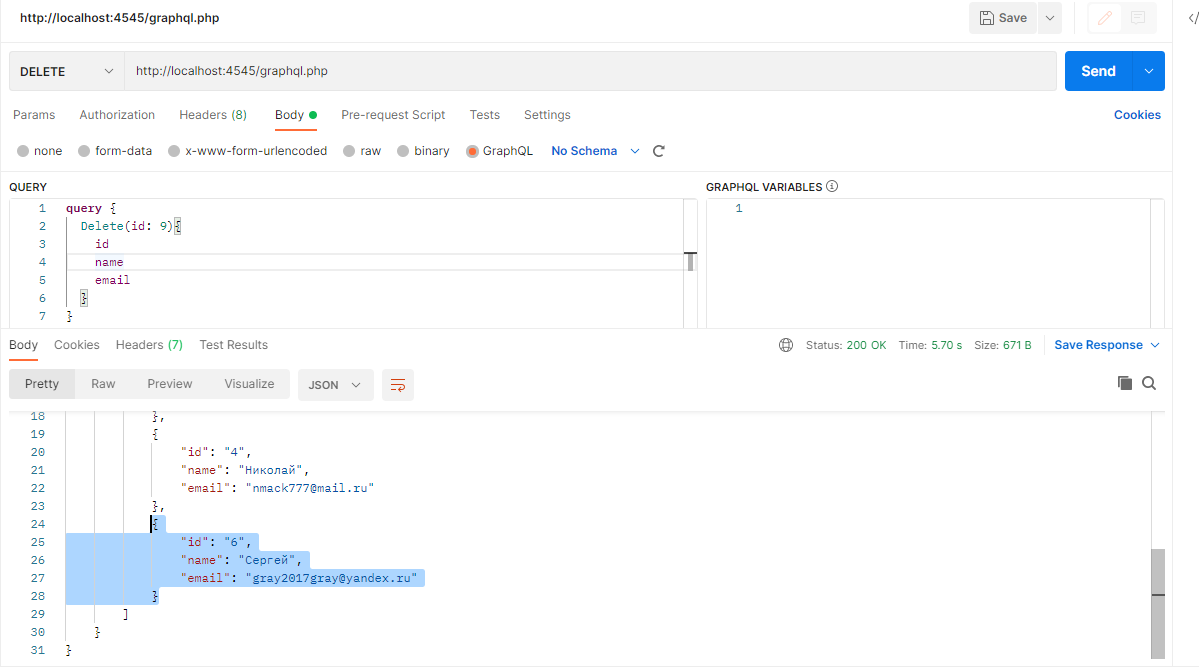


Рис.4-проверка запроса

## Вывод:

В ходе выполнения был создан api.

## Вопросы:

1. Что такое HTTP-запрос?

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

2. Опишите существующие HTTP-запросы

Особенности GET запроса:

-может быть закэширован

-остается в истории браузера

-может быть закладкой в браузере

-не должен использоваться при работе с крайне -важными данными

-имеет ограниченную длину

-должен применяться только для получения данных (ред.)

Особенности POST запроса:

-не кэшируется

-не может быть закладкой в браузере

-не остаётся в истории браузера

-нет ограничений по длине запроса

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. Таким образом, если пара first\_name = Nina передана методом GET , то $HTTP\_GET\_VARS["first\_name"]="Nina".

Использовать в программе имена переданных переменных напрямую небезопасно. Поэтому было решено начиная с PHP 4.1.0 задействовать для обращения к переменным, переданным с помощью HTTP-запросов, специальный массив – $\_REQUEST . Этот массив содержит данные, переданные методами POST и GET , а также с помощью HTTP cookies. Это суперглобальный ассоциативный массив, т.е. его значения можно получить в любом месте программы, используя в качестве ключа имя соответствующей переменной (элемента формы).

4. Опишите создание HTML-форм на PHP.

Вставка формы осуществляется напрямую в HTML—код страницы. Главный элемент формы называется <form>. Уже внутрь него добавляются все остальные элементы – текстовые поля, «чекбоксы», переключатели и т.д. У элемента <form> имеется несколько атрибутов, один из которых является обязательным. Он называется action. В action указывается, где именно будет приниматься и обрабатываться информация, переданная посредством формы. Как правило, обработка происходит в стороннем PHP—файле. Пример использования атрибута– action=»obrabotchik.php«. Атрибут method позволяет задать метод передачи информации. По умолчанию (если не прописывать атрибут) будет указан метод GET. В данном случае информация передается напрямую через URL—адрес. Для каждого элемента формы будет создана пара следующего вида – «имя элемента = значение, которое в нем лежит». Все эти пары, разделенные знаком «амперсанд» будут перечислены в адресной строке. Если прописать method=»POST» (регистр не важен), то данные будут передаваться не через URL, а через тело запроса (в скрытом режиме)

5. Что такое API?

API — описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой. Обычно входит в описание какого-либо интернет-протокола, программного каркаса или стандарта вызовов функций операционной системы.

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. При этом обычно компоненты образуют иерархию — высокоуровневые компоненты используют API низкоуровневых компонентов, а те, в свою очередь, используют API ещё более низкоуровневых компонентов. По такому принципу построены протоколы передачи данных по Интернету. Стандартный стек протоколов (сетевая модель OSI) содержит 7 уровней (от физического уровня передачи бит до уровня протоколов приложений, подобных протоколам HTTP и IMAP). Каждый уровень пользуется функциональностью предыдущего («нижележащего») уровня передачи данных и, в свою очередь, предоставляет нужную функциональность следующему («вышележащему») уровню.

Понятие протокола близко по смыслу к понятию API. И то, и другое является абстракцией функциональности, только в первом случае речь идёт о передаче данных, а во втором — о взаимодействии приложений.

API библиотеки функций и классов включает в себя описание сигнатур и семантики функций.

7. Что такое Web API?

Это интерфейс прикладного программирования для веб-сервера или веб-браузера. Это концепция веб-разработки, обычно ограниченная клиентской стороной веб-приложения, 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.

Каждый раз, когда пользователь посещает какую-либо страницу в сети, он взаимодействует с API удалённого сервера. API — это составляющая часть сервера, которая получает запросы и отправляет ответы.

9. Что такое REST?

Это архитектура, т.е. принципы построения распределенных гипермедиа систем, того что другими словами называется World Wide Web, включая универсальные способы обработки и передачи состояний ресурсов по HTTP.

10.Как организована передача данных в архитектуре REST?

Архитектура REST требует соблюдения следующего условия. В период между запросами серверу не нужно хранить информацию о состоянии клиента и наоборот. Все запросы от клиента должны быть составлены так, чтобы сервер получил всю необходимую информацию для выполнения запроса. Таким образом и сервер, и клиент могут «понимать» любое принятое сообщение, не опираясь при этом на предыдущие сообщения.

11.Как организована работа REST?

Это набор принципов и ограничений взаимодействия клиента и сервера в сети интернет, использующий существующие стандарты (HTTP протокол, стандарт построения URL, форматы данных JSON и XML) в ходе взаимодействии

12.Что такое SOAP?

SOAP (Simple Object Access Protocol) — стандартный протокол по версии W3C.

13.Чем SOAP отличается от REST?

· SOAP обозначает простой протокол доступа к объектам, тогда как REST обозначает передачу представительного состояния.

· SOAP — это протокол, тогда как REST — это архитектурный паттерн.

· SOAP использует сервисные интерфейсы для предоставления своих функций клиентским приложениям, а REST использует унифицированные локаторы сервисов для доступа к компонентам на аппаратном устройстве.

· SOAP требует большей пропускной способности для его использования, тогда как REST не требует большой пропускной способности.

· SOAP работает только с форматами XML, тогда как REST работает с простым текстом, XML, HTML и JSON.

· SOAP не может использовать REST, тогда как REST может использовать SOAP.

14.Для чего нужен SOAP-процессор?

SOAP основан на языке XML и расширяет некоторый протокол прикладного уровня — HTTP, FTP, SMTP и т.д. Как правило чаще всего используется HTTP. Вместо использования HTTP для запроса HTML-страницы, которая будет показана в браузере, SOAP отправляет посредством HTTP-запроса XML-сообщение и получает результат в HTTP-отклике. 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 состоит из заголовка — элемент SOAP-ENV:Header и основной части — элемент SOAP-ENV:Body. Заголовок может содержать метаданные, относящиеся к сообщению в целом. В теле сообщения передается элемент params с входными параметрами метода. Формат элемента params отличается для разных методов.

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

Является самым «верхним» элементом SOAP сообщения. Содержит корневой элемент

XML-документа. Описывается с помощью элемента Envelope с обязательным

пространством имен http://www.w3.org/2003/05/soap-envelope для версии 1.2 и

http://schemas.xmlsoap.org/soap/ для версии 1.1.

У элемента Envelope могут быть атрибуты xmlns, определяющие пространства

имен, и другие атрибуты, снабженные префиксами.

Envelope может иметь необязательный дочерний элемент Header c тем же

пространством имен — заголовок. Если этот элемент присутствует, то он должен быть

3

Рисунок 2. Структура

SOAP сообщения

Сервис-ориентированные технологии интеграции информации. Автор - Фастовский Э.Г. 2011 г.

первым прямым дочерним элементом конверта.

Следующий дочерний элемент конверта должен иметь имя Body и то же самое

пространство имен - тело. Это обязательный элемент и он должен быть вторым прямым

дочерним элементом конверта, если есть заголовок, или первым — если заголовка нет.

Версия 1.1 позволяла после тела сообщения записывать произвольные элементы,

снабженные префиксами. Версия 1.2 это запрещает.

Элементы Header и Body могут содержать элементы из различных пространств

имен.

Конверт изменяется от версии к версии. SOAP-процессоры, совместимые с версией

1.1, при получении сообщения, содержащего конверт с пространством имен версии 1.2,

будут генерировать сообщение об ошибке. Аналогично для SOAP-процессоров, совместимых

с версией 1.2. Ошибка — VersionMismatch.

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

Заголовок может содержать метаданные, относящиеся к сообщению в целом.Заголовок содержит элемент locale, устанавливающий русский язык для ответных сообщений, и элемент token — авторизационный токен.

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

Тело SOAP-сообщения является обязательным элементом внутри env:Envelope, содержащим основную информацию SOAP-сообщения, которая должна быть передана из начальной точки пути сообщения в конечную

19.Опишите SOAP-сообщение с вложением.

SOAP-сообщение представляет собой XML-документ; сообщение состоит из трех

основных элементов: конверт (SOAP Envelope), заголовок (SOAP Header) и тело (SOAP

Body).

20.Что такое graphql?

Язык запросов и обработки данных с открытым исходным кодом для API и среда выполнения для выполнения запросов с существующими данными.

21.Что такое Распознаватели (resolvers) в graphql?

Resolver или распознаватель — функция, которая возвращает данные для определённого поля. Resolver’ы возвращают данные того типа, который определён в схеме. Распознаватели могут быть асинхронными.

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

Она состоит из двух взаимосвязанных объектов: TypeDefs и Resolvers. Выше были описаны основные типы GraphQL. Чтобы сервер мог с ними работать, эти типы необходимо определить. Объект typeDef определяет список типов, которые доступны в проекте.

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

Валидация – это проверка продукта, процесса или системы на соответствие требованиям клиента.

24.Где и когда выполнять валидацию данных?

Валидация проводится тогда, когда невозможно оценить соответствие продукта, процесса или системы требованиям клиента до того, как клиент начнет этим продуктом пользоваться. Например, если речь идет о программном обеспечении, в него встраивается валидационный код. Этот код клиент вводит, если продукт полностью соответствует его ожиданиям и выполняет нужные задачи. В противном случае доступ к продукту прекращается и проводятся его доработки либо исполнитель возвращает деньги.

25.Как выполнять валидацию данных?

Перед отправкой данных на сервер важно убедиться, что все обязательные поля формы заполнены данными в корректном формате. Это называется валидацией на стороне клиента и помогает убедиться, что данные, введённые в каждый элемент формы, соответствуют требованиям.

26.Приведите пример с поэтапной валидацией данных.

Регистрация и авторизация: пользователь вводит валидные данные email и валидные данные пароль

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

К первому виду относятся запросы на чтение данных, которые в терминологии GraphQL называются просто запросами (query) и относятся к букве R (reading, чтение) акронима CRUD. Запросы второго вида — это запросы на изменение данных, которые в GraphQL называют мутациями (mutation).

## Список использованной литературы:

1. Конспект лекций по дисциплине «Разработка сервисных частей интернет ресурсов», РТУ МИРЭА, лектор –преподаватель Куликов А.А.

2. Docker – URL: https://www.docker.com/ (Дата обращения – 17.10.2021)

3. Настройка стека ламп в контейнерах Docker– URL: https://www.centlinux.com/2020/03/configure-lamp-stack-in-docker-containers.html (Дата обращения – 17.10.2021)

4. Полное практическое руководство Docker – URL: https://habr.com/ru/post/310460/ (Дата обращения – 17.10.2021)