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-03-20 Постнов Н.С.

_____
(подпись студента)

Руководитель практической работы преподаватель Волков М.Ю.

_____
(подпись руководителя)

Работа представлена «____»_____ 2022 г.

Допущен к работе «___» _____ 2022 г.

Москва 2022

СОДЕРЖАНИЕ

1. Цель работы

Вариант 6 - библиотека

Технические требования к реализации интерфейса:

- Доступ как минимум к 2 независимым сущностям.
- Реализация как минимум операций группы CRUD(создание, чтение, обновление, удаление). Приветствуется реализация дополнительной функциональности.
- Тестирование всех функциональных возможностей созданного интерфейса с использованием программы Postman.

2. Ход работы

Создам файл сущности, в котором будет описана логика выполнения каждой над ним операции (рис. 1).

```php
<?php
class Books{

    private $ordersTable = "books";
    public $id;
    public $title;
    public $author;
    public $taken;
    private $conn;

    public function __construct($db){
        $this->conn = $db;
    }

    function read(){
        if($this->id) {
            $stmt = $this->conn->prepare("SELECT * FROM ".$this->ordersTable." WHERE ID = ?");

            $stmt->bind_param("i", $this->id);



        } else {
            $stmt = $this->conn->prepare("SELECT * FROM ".$this->ordersTable);
        }
        $stmt->execute();
        $result = $stmt->get_result();
        return $result;
    }
```

Рисунок 1 – Сущность книги

Далее реализую операции группы CRUD (рис. 2-5):

```php
<?php

header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Methods: POST");
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization,
X-Requested-With");
```

```php
include_once '/home/public_html/database/database.php';
include_once '/home/public_html/class/Books.php';

error_reporting(E_ALL);
ini_set('display_errors', 1);

$database = new Database();
$db = $database->getConnection();

$items = new Books($db);

ini_set("allow_url_fopen", true);



$data = json_decode(file_get_contents('php://input'));

if(!empty($data->author)){

    $items->title = $data->title;
    $items->author = $data->author;
    $items->taken = $data->taken;

    if ($items->create()){
        http_response_code(201);
        echo json_encode(array("message" => "Item was created."));
    }
    else{

        http_response_code(503);
        echo json_encode(array("message" => "Unable to create item."));
    }
}else{
    echo "here1";
    http_response_code(400);
    echo json_encode(array("message" => "Unable to create item. Data is incomplete."));
}
```

Рисунок 2 – Операция создания

```php
<?php
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");

include_once '/home/public_html/database/database.php';
include_once '/home/public_html/class/Books.php';
```

```php
$database = new Database();
$db = $database->getConnection();

$items = new Books($db);

$items->id = (isset($_GET['id']) && $_GET['id']) ? $_GET['id'] : '0';

$result = $items->read();

if($result->num_rows > 0){
    $itemRecords=array();
    $itemRecords["items"]=array();
    while ($item = $result->fetch_assoc()) {
        extract($item);
        $itemDetails=array(
            "id" => $ID,
            "title" => $title,
            "author" => $author,
            "taken" => $taken
        );
       array_push($itemRecords["items"], $itemDetails);
    }
    http_response_code(200);
    echo json_encode($itemRecords);
}else{
    http_response_code(404);
    echo json_encode(
        array("message" => "No item found.")
    );
}
```

Рисунок 3 – Операция чтения

```php
<?php
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Methods: POST");
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization,
X-Requested-With");

include_once '/home/public_html/database/database.php';
include_once '/home/public_html/class/Books.php';
```

```php
$database = new Database();
$db = $database->getConnection();

$items = new Books($db);

$data = json_decode(file_get_contents("php://input"));

if(!empty($data->author)){

    $items->id = $data->id;
    $items->title = $data->title;
    $items->author = $data->author;
    $items->taken = $data->taken;

    if($items->update()){
        http_response_code(200);
        echo json_encode(array("message" => "Item was updated."));
    }else{
        http_response_code(503);
        echo json_encode(array("message" => "Unable to update items."));
    }

} else {
    http_response_code(400);
    echo json_encode(array("message" => "Unable to update items. Data is incomplete."));
}
```

Рисунок 4 – Операция обновления

```php
<?php
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Methods: POST");
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization,
X-Requested-With");

include_once '/home/public_html/database/database.php';
include_once '/home/public_html/class/Books.php';

$database = new Database();
$db = $database->getConnection();

$items = new Books($db);
```

```php
$data = json_decode(file_get_contents("php://input"));

if(!empty($data->id)) {
    $items->id = $data->id;
    if($items->delete()){
        http_response_code(200);
        echo json_encode(array("message" => "Item was deleted."));
    } else {
        http_response_code(503);
        echo json_encode(array("message" => "Unable to delete item."));
    }
} else {
    http_response_code(400);
    echo json_encode(array("message" => "Unable to delete items. Data is incomplete."));
}
```

Рисунок 5 – Операция удаления

Спецификация API:

Книги

POST: /admin/api/create.php

GET: /admin/api/read.php?{id}

DELETE: /admin/api/delete.php

UPDATE: /admin/api/update.php

Заказы

POST: /api/create.php

GET: /api/read.php?{id}

DELETE: /api/delete.php

UPDATE: /api/update.php

Далее приведены скриншоты работы программы в Postman

Создание экземпляра книги (рис. 6)

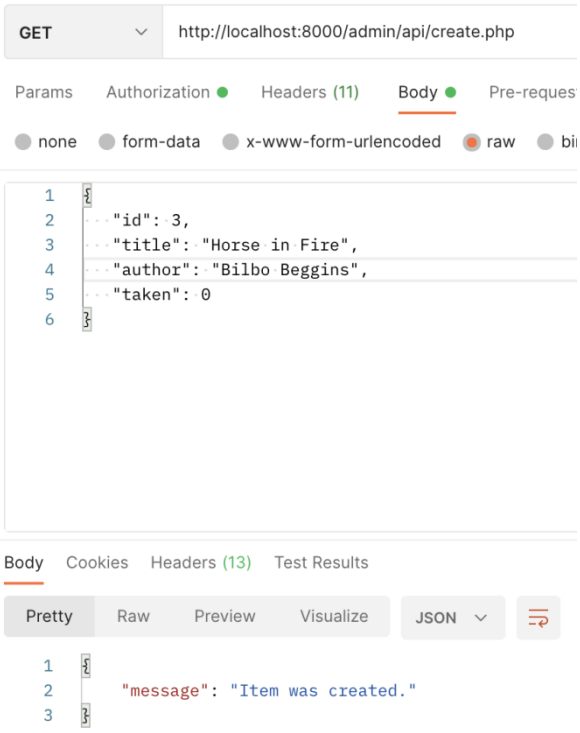

Рисунок 6 – Создание экземпляра книги

Чтение экземпляров книги (рис. 7)

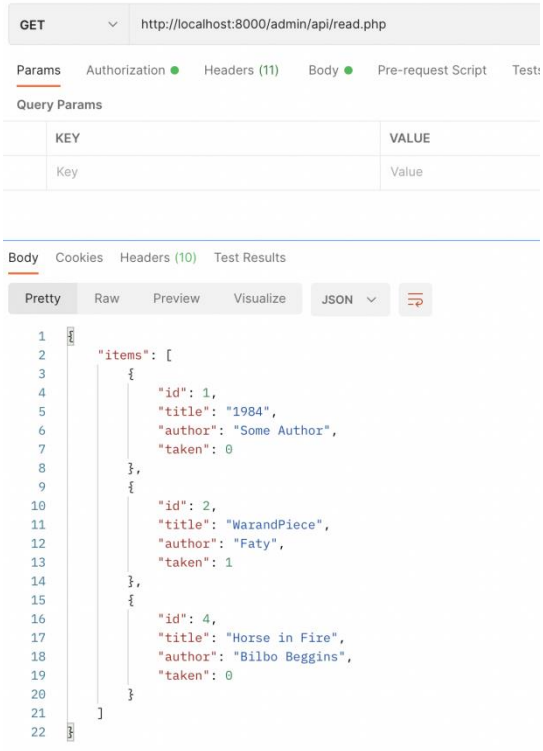

Рисунок 7 – Чтение экземпляров книги

Обновление экземпляра книги (рис. 8)

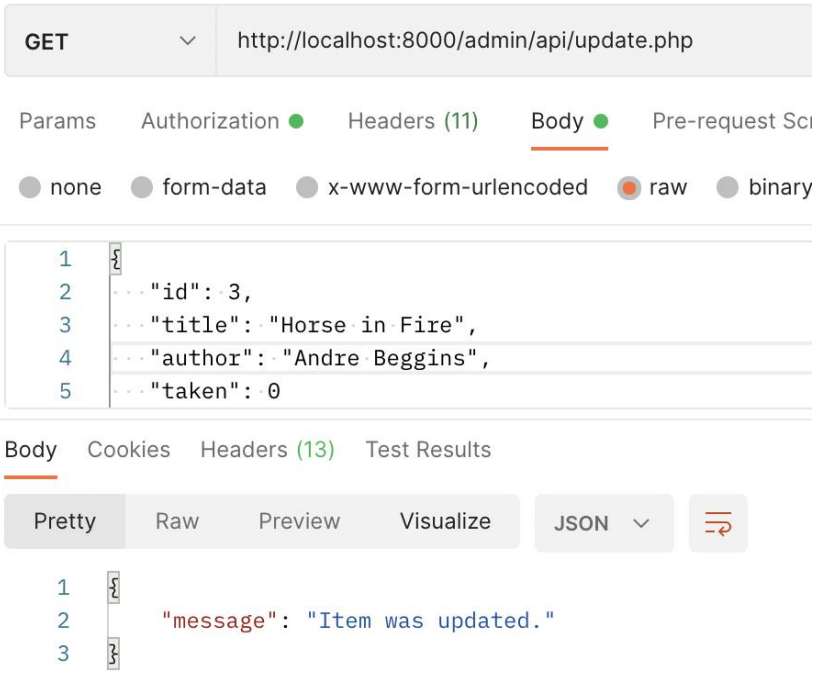

Рисунок 8 – Обновление экземпляра книги

Удаление экземпляра книги (рис. 9)

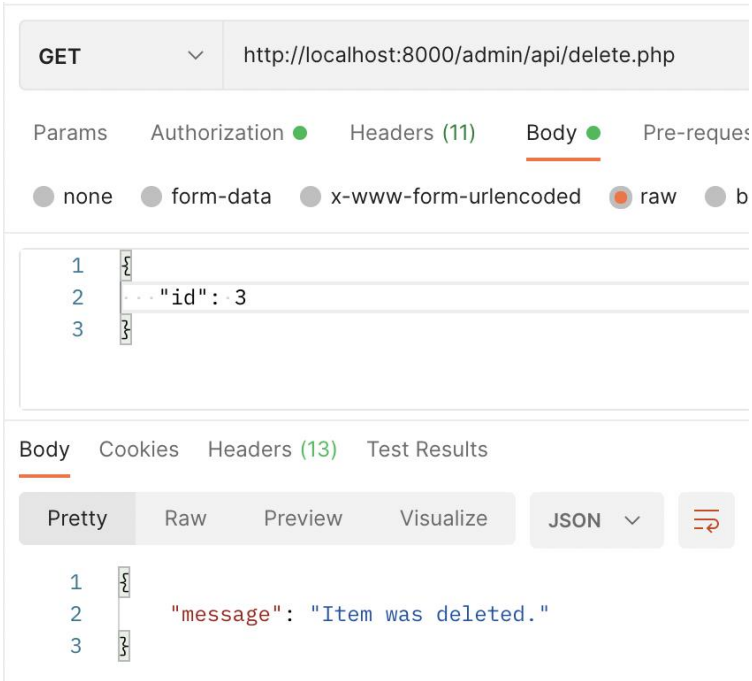

Рисунок 9 – Удаление экземпляра книги

А также соответственные скриншоты работы для сущности заказа:

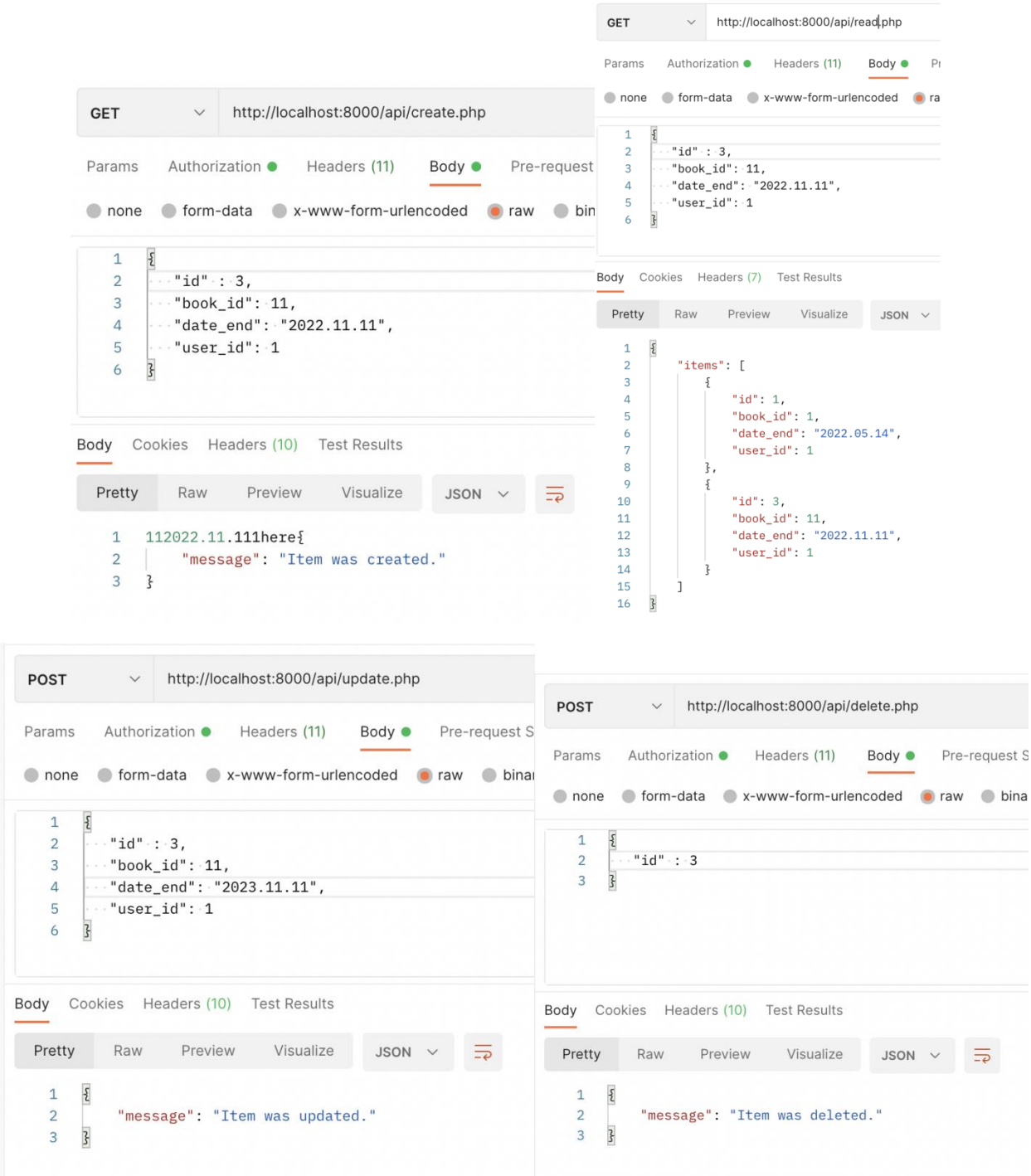

Рисунок 10-14 – CRUD с сущностью заказа

3. Выводы

В ходе работы была реализована спецификация CRUD:

1. Создание сущности;
2. Удаление сущности;
3. Обновление сущности;
4. Чтение сущности.

Реализация была воспроизведена на основе базы данных mysql. Тестирование верности работы программного модуля было обеспечено с помощью программного обеспечения Postman

4. Ответы на вопросы к практической работе

1. Что такое HTTP-запрос?

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

HTTP-запрос состоит из трех элементов:

- стартовой строки, которая задает параметры запроса или ответа
- заголовка, который описывает сведения о передаче и другую служебную информацию
- тело (его не всегда можно встретить в структуре). Обычно в нем как раз лежат передаваемые данные. От заголовка тело отделяется пустой строкой

2. Опишите существующие HTTP-запросы.

Для разграничения действий с ресурсами на уровне HTTP-методов и были придуманы следующие варианты:

- GET — запрашивает представление ресурса. Запросы с использованием этого метода могут только извлекать данные.
- HEAD — запрашивает ресурс так же, как и метод GET, но без тела ответа.
- POST — используется для отправки сущностей к определённому ресурсу. Часто вызывает изменение состояния или какие-то побочные эффекты на сервере.
- PUT — заменяет все текущие представления ресурса данными запроса.
- DELETE — удаляет указанный ресурс.
- CONNECT — устанавливает "туннель" к серверу, определённому по ресурсу.
- OPTIONS — используется для описания параметров соединения с

ресурсом.

- TRACE — выполняет вызов возвращаемого тестового сообщения с ресурса.
- PATCH — используется для частичного изменения ресурса.

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

Любой запрос клиента к серверу должен начинаться с указания метода. Метод сообщает о цели запроса клиента. Протокол HTTP поддерживает достаточно много методов, но реально используются только три: POST , GET и HEAD . Метод GET позволяет получить любые данные, идентифицированные с помощью URL в запросе ресурса. Если URL указывает на программу, то возвращается результат работы программы, а не ее текст (если, конечно, текст не есть результат ее работы). Дополнительная информация, необходимая для обработки запроса, встраивается в сам запрос (в адресную строку). При использовании метода GET в поле тела ресурса возвращается собственно затребованная информация (текст HTML-документа, например).

4. Опишите создание HTML-форм на PHP.

PHP содержит множество средств для работы с формами. Это позволяет очень просто решать типичные задачи, которые часто возникают в веб-программировании:

- Регистрация и аутентификация пользователя;
- Отправка комментариев на форумах и социальных сетях;
- Оформление заказов.

Общий принцип действия функционала: сначала пользователь регистрируется через соответствующую форму. После заполнения всех полей данные отправляются для обработки на сервер, где заносятся в таблицу БД. При

следующем заходе на ресурс юзер вводит указанные при регистрации логин и пароль. Их правильность проверяется путем выборки данных из таблицы. Если оба значения указаны правильно, то пользователь попадает на страницу приветствия. Иначе выдается сообщение о неправильном вводе пароля и логина.

5. Что такое API?

Популярный термин API (англ. Application Programming Interface — программный интерфейс приложения) — это набор способов и правил, по которым различные программы общаются между собой и обмениваются данными.

Все эти коммуникации происходят с помощью функций, классов, методов, структур, а иногда констант одной программы, к которым могут обращаться другие. Это основной принцип работы API.

6. Опишите API как средство интеграции приложений.

Интеграция API – это соединение между двумя или более приложениями через их API (интерфейсы прикладного программирования), которые позволяют системам обмениваться источниками данных. Интеграция API позволяет управлять процессами во многих секторах и уровнях организации, обеспечивая синхронизацию данных, повышая производительность и увеличивая прибыль.

7. Что такое Web API?

Web API или Web Service API –это интерфейс обработки приложений между веб-сервером и веб-браузером. Все веб-сервисы являются API, но не все API являются веб-сервисами. REST API – это особый тип Web API, в котором используется стандартный архитектурный стиль, описанный выше.

8. Приведите пример API.

На практике API могут использоваться для связи практически любых процессов. Вот несколько распространенных примеров использования API:

- Обмен информацией о рейсах между авиакомпаниями и туристическими сайтами
- Использование Google Maps в приложении для совместных поездок (райдшеринга)
- Создание виртуальных собеседников в службе обмена сообщениями
- Встраивание видеоклипов с YouTube на веб-странице
- Автоматизация рабочих процессов в программных инструментах для B2B-сектора

9. Что такое REST?

REST (от англ. Representational State Transfer — «передача репрезентативного состояния» или «передача „самоописываемого“ состояния») — 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?

В REST архитектуре клиенты отправляют на сервер запросы для получения или модификации данных, а сервера отправляют клиентам ответы на их запросы.

11. Как организована работа REST?

Как было сказано выше, REST определяет, как компоненты распределенной системы должны взаимодействовать друг с другом. В общем случае этот происходит посредством запросов-ответов. Компоненту, которая отправляет запрос называют клиентом; компоненту, которая обрабатывает запрос и отправляет клиенту ответ, называют сервером. Запросы и ответы, чаще всего, отправляются по протоколу HTTP (англ. HyperText Transfer Protocol — «протокол передачи гипертекста»). Как правило сервер — это некое веб-

приложение. Клиентом же может быть не то чтобы что угодно, но довольно многое. Например, мобильное приложение, которое запрашивает у сервера данные. Либо браузер, который отправляет запросы с веб-страницы на сервер для загрузки данных. Приложение А может запрашивать данные у приложения Б. Тогда А является клиентом по отношению к Б, а Б — сервером по отношению к А. Одновременно с этим, А может обрабатывать запросы от В, Г, Д и т.д. В таком случае, приложение А является одновременно и сервером, и клиентом. Все зависит от контекста. Однозначно одно: компонента которая шлет запрос — это клиент. Компонента, которая принимает, обрабатывает и отвечает на запрос — сервер.

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?

REST был создан для решения проблем SOAP. Поэтому у него более гибкая архитектура. Он состоит только из простых рекомендаций и позволяет разработчикам реализовывать рекомендации по-своему. Он допускает различные форматы сообщений, такие как HTML, JSON, XML и простой текст, в то время как SOAP допускает только XML. REST также является более легкой архитектурой, поэтому веб-сервисы RESTful имеют более высокую производительность.

14. Для чего нужен SOAP-процессор?

SOAP позволяет разработчикам вызывать процессы, запущенные в разных операционных системах (таких как Windows, macOS и Linux), для

аутентификации, авторизации и обмена данными с использованием расширяемого языка разметки (XML). Поскольку веб-протоколы, такие как HTTP, установлены и работают практически во всех операционных системах, SOAP позволяет клиентам вызывать веб-службы и получать ответы независимо от языка и платформы.

15. Опишите общую структуру SOAP-сообщения.

SOAP-сообщение – это обычный XML-документ, содержащий следующие элементы:

- Конверт
- Заголовок
- Тело
- Неисправность

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

Конверт SOAP аналогичен конверту обычного письма. Он содержит информацию о письме, которое будет шифровано в основном разделе SOAP, включая данные о получателе и отправителе, а также информация о самом сообщении. Например, заголовок конверта SOAP может указывать на то, как должно обрабатываться сообщение.

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

Заголовок – содержит любые необязательные атрибуты сообщения, используемые при обработке сообщения, либо в промежуточной точке, либо в конечной конечной точке. Это необязательный элемент.

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

Тело – содержит данные XML, содержащие отправляемое сообщение. Это обязательный элемент.

19. Опишите SOAP-сообщение с вложением.

Неисправность – необязательный элемент неисправности, который предоставляет информацию об ошибках, возникающих при обработке сообщения.

20. Что такое graphql?

GraphQL — это язык запросов для API - интерфейсов и среда, в которой они выполняются. С помощью GraphQL можно получить данные из API и передать их в приложение (от сервера к клиенту). Официальная документация. GraphQL есть только на английском языке, на русский язык пока ещё не переведена.

21. Что такое Распознаватели (resolvers) в graphql?

Резолверы — это функции, которые запускаются каждый раз, когда запрос запрашивает поле. Когда реализация GraphQL получает запрос, она выполняет резолвер для каждого поля. Если резолвер возвращает поле типа Object, то GraphQL запускает резолвер-функцию этого поля. Когда все резолверы возвращают скалярные значения, цепочка замыкается, и запрос получает готовый JSON-результат.

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

GraphQL используется для работы с данными в «вашем приложении», а не «в вашей базе данных». Дело в том, что GraphQL — это система, независимая от источников данных, то есть, для организации её работы неважно — где именно хранятся данные.

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. Валидация данных проводится, как правило, после выполнения операций ETL и для подтверждения корректности результатов работы моделей

машинного обучения (предиктов).

24.Где и когда выполнять валидацию данных?

Для валидации требуется доступ к недоступной части состояния системы. Это особенно характерно для проверки данных, вводимых человеком через графический интерфейс пользователя. Современные приложения часто построены с использованием многоуровневой архитектуры, которая предполагает, что реализация пользовательского интерфейса выделена в презентационный слой, а для проверки требуется доступ к другим слоям, вплоть до слоя базы данных.

Валидация требует полностью повторить логику обработки. Как уже отмечено двумя абзацами выше, при многослойной архитектуре приложения пользовательский интерфейс обычно выделяется в специальный презентационный слой, а логика обработки данных находится на другом слое. И бывают такие ситуации, когда для валидации нужно практически полностью выполнить эту обработку, потому не существует более короткого способа понять, завершится она успехом или нет.

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».

Проверка отдельных значений. Для пользовательского интерфейса это проверка значения в отдельном поле, причём выполняться она может как по мере ввода (проверяется то неполное значение, которое введено к настоящему моменту), так и после завершения ввода, когда поле теряет фокус. Для программного интерфейса (API) это проверка одного из параметров, переданных в вызываемую процедуру. Для данных, получаемых из файла, это проверка какого-то

прочитанного фрагмента файла. Такие проверки, опять-таки по аналогии с компиляторной терминологией, можно назвать «синтаксическими».

Совокупность входных значений. Можно предположить, что в программу сначала передаются какие-то данные, после чего подаётся некоторый сигнал, который инициирует их обработку. Например, пользователь ввёл данные в форму или в несколько форм (в так называемом «визарде») и наконец нажал кнопку «OK». В этот момент можно выполнить так называемые «семантические» проверки, нацеленные на валидацию не только отдельных значений, но и взаимосвязей между ними, взаимных ограничений.

Проверка состояния системы после обработки данных. Наконец, есть последний способ, к которому можно прибегнуть, если валидацию непосредственно входных данных выполнить не удаётся — можно попытаться их обработать, но оставить возможность вернуть всё к исходному состоянию. Такой механизм часто называется транзакционным.

26.Приведите пример с поэтапной валидацией данных.

Рассмотрим пример розничного продавца, который собирает данные о своих магазинах, но не может создать надлежащую проверку почтового индекса. Надзор может затруднить использование данных для получения информации и бизнес-аналитики. Если почтовый индекс не введен или введен неправильно, может возникнуть несколько проблем.

В некоторых картографических программах может быть сложно определить местоположение хранилища. Почтовый индекс магазина также поможет получить представление о районе, в котором расположен магазин. Без проверки данных почтового индекса более вероятно, что данные потеряют ценность. Это приведет к дополнительным расходам, если потребуется восстановить данные или ввести почтовый индекс вручную.

Простым решением этой проблемы было бы установить флажок, гарантирующий ввод действительного почтового индекса. Решением может быть выпадающее меню или форма автоматического заполнения, которая позволяет пользователю выбрать почтовый индекс из списка действительных кодов. Такой тип проверки данных называется проверкой кода или проверкой кода.

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

В GraphQL есть только два типа операций, которые вы можете выполнять: запросы и мутации.

В то время как мы используем запросы для извлечения данных, мы используем мутации для изменения данных на стороне сервера.

Если запросы в GraphQL эквивалентны вызовам GET в REST, то мутации представляют собой методы изменения состояния в REST (например, DELETE, PUT, PATCH и т.д.).

5. **Ссылка на удалённый репозиторий проекта**

https://github.com/caprize/university_backend

6. **Список использованных источников**

1. Видео “Введение в Докер” на английском языке от создателя: Introduction to Docker (https://www.youtube.com/watch?v=Q5POuMHxW-0)
2. Статья о назначении докера простыми словами: https://habr.com/ru/post/309556/
3. Более сложная и подробная статья про докер: https://habr.com/ru/post/277699/