|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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 № 3**

по дисциплине «Разработка серверных частей интернет-ресурсов»

**Тема практической работы:** Создание REST API на языке PHP

|  |  |
| --- | --- |
| Студент группы ИКБО-32-21 | Быченков А.К. |

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

(подпись студента)

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

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

(подпись руководителя)

Работа представлена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Москва 2023

**Цель работы:** получить навыки работы с базой данных и создания REST API.

**Задание**

Вам необходимо создать конфигурацию docker-compose для вашего веб-сервера или взять из предыдущей практики.

Docker контейнер веб сервер должен включать:

* ОС Linux
* Веб-сервер Apache

Docker контейнер базы данных должен включать:

* ОС Linux
* Базу данных Mysql

Docker-compose:

* Docker-контейнеры, созданные ранее
* Примонтированные тома
* Настройки портов
* Связь между контейнерами
* Переменные внешнего окружения
* Настройки базы данных

1. Создать SQL скрипт инициализации, где студенту необходимо реализовать любые две связанные сущности.

2. Требуется реализовать REST API для двух сущностей, где на запросы ответы должны быть в формате JSON.

Пример запроса: `[GET] localhost:8080/api/orders.php?id=3`

3. Провести ручное тестирование с помощью POSTMAN, CURL или Insomnia.

**Ход работы**

Был написан sql-скрипт инициализации для создания двух связанных сущностей:

Листинг 1 – init.sql

CREATE DATABASE IF NOT EXISTS appDB;

USE appDB;

CREATE TABLE IF NOT EXISTS orders (

id INT(11) NOT NULL AUTO\_INCREMENT,

address VARCHAR(100) NOT NULL,

customer\_name VARCHAR(63) NOT NULL,

status VARCHAR(15) NOT NULL DEFAULT 'new',

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (id)

);

CREATE TABLE IF NOT EXISTS goods (

id INT(11) NOT NULL AUTO\_INCREMENT,

title VARCHAR(63) NOT NULL,

description TEXT NOT NULL,

price DECIMAL(10,2) NOT NULL,

PRIMARY KEY (id)

);

CREATE TABLE IF NOT EXISTS ordered\_goods (

id INT(11) NOT NULL AUTO\_INCREMENT,

order\_id INT(11),

good\_id INT(11),

quantity INT(11) NOT NULL,

PRIMARY KEY (id),

CONSTRAINT orders\_id\_fk FOREIGN KEY (order\_id) REFERENCES orders(id) ON DELETE SET NULL,

CONSTRAINT goods\_id\_fk FOREIGN KEY (good\_id) REFERENCES goods(id) ON DELETE SET NULL

);

Были написан слой взаимодействия с базой данных для созданных сущностей:

Листинг 2 – goodRepository.php

<?php

include 'repository/db.php';

include 'dto/goodCreate.php';

class GoodRepository

{

public static function getAll(): array

{

global $mysqli;

$result = $mysqli->query(

'SELECT \* FROM goods'

)->fetch\_all(MYSQLI\_ASSOC);

return array\_map(function ($good) {

$good["id"] = intval($good["id"]);

return $good;

}, $result);

}

public static function getById(int $id): ?array

{

global $mysqli;

$result = $mysqli->query(

"SELECT \* FROM goods WHERE id='$id'"

)->fetch\_assoc();

if ($result) {

$result["id"] = intval($result["id"]);

return $result;

}

return null;

}

public static function create(GoodCreate $good): ?int

{

global $mysqli;

$mysqli->query(

"INSERT INTO goods (title, description, price) VALUES ('$good->title', '$good->description', '$good->price')"

);

return $mysqli->insert\_id;

}

public static function update(int $id, GoodCreate $good): ?int

{

global $mysqli;

$mysqli->query("INSERT INTO goods (id, title, description, price) VALUES ('$id', '$good->title', '$good->description', '$good->price') ON DUPLICATE KEY UPDATE title='$good->title', description='$good->description', price='$good->price'");

return $id;

}

public static function partialUpdate(int $id, GoodCreate $good): ?int

{

global $mysqli;

if ($good->title)

$mysqli->query(

"UPDATE goods SET title='$good->title' WHERE id='$id'"

);

if ($good->description)

$mysqli->query(

"UPDATE goods SET description='$good->description' WHERE id='$id'"

);

if ($good->price)

$mysqli->query(

"UPDATE goods SET price='$good->price' WHERE id='$id'"

);

return $id;

}

public static function delete(int $id): bool

{

global $mysqli;

$mysqli->query(

"DELETE FROM goods WHERE id='$id'"

);

return $mysqli->affected\_rows > 0;

}

}

Листинг 3 – orderRepository.php

<?php

include 'repository/db.php';

include 'dto/orderCreate.php';

class OrderRepository

{

public static function getAll(): array

{

global $mysqli;

$result = $mysqli->query(

'SELECT orders.id, orders.address, orders.customer\_name, orders.status, orders.created\_at,

JSON\_ARRAYAGG(JSON\_OBJECT(

"id", goods.id,

"title", goods.title,

"description", goods.description,

"price", goods.price,

"quantity", ordered\_goods.quantity)) as goods

FROM orders

LEFT JOIN ordered\_goods ON orders.id = ordered\_goods.order\_id

LEFT JOIN goods on ordered\_goods.good\_id = goods.id

GROUP BY orders.id'

)->fetch\_all(MYSQLI\_ASSOC);

return array\_map(function ($order) {

$order["id"] = intval($order["id"]);

$order['goods'] = json\_decode($order['goods'], true);

return $order;

}, $result);

}

public static function getById(int $id): ?array

{

global $mysqli;

$result = $mysqli->query(

"SELECT orders.id, orders.address, orders.customer\_name, orders.status, orders.created\_at,

JSON\_ARRAYAGG(JSON\_OBJECT(

'id', goods.id,

'title', goods.title,

'description', goods.description,

'price', goods.price,

'quantity', ordered\_goods.quantity)) as goods

FROM orders

LEFT JOIN ordered\_goods ON orders.id = ordered\_goods.order\_id

LEFT JOIN goods on ordered\_goods.good\_id = goods.id

WHERE orders.id='$id'

GROUP BY orders.id"

)->fetch\_assoc();

if ($result) {

$result["id"] = intval($result["id"]);

$result['goods'] = json\_decode($result['goods'], true);

return $result;

}

return null;

}

public static function create(OrderCreate $order): ?int

{

global $mysqli;

$mysqli->query("INSERT INTO orders (address, customer\_name) VALUES ('$order->address', '$order->customer\_name')");

$order\_id = $mysqli->insert\_id;

foreach ($order->goods as $good) {

$good\_id = $good['id'];

$quantity = $good['quantity'];

$mysqli->query("INSERT INTO ordered\_goods (order\_id, good\_id, quantity) VALUES ('$order\_id', '$good\_id', '$quantity')");

}

return $order\_id;

}

public static function update(int $id, OrderCreate $order): ?int

{

global $mysqli;

$mysqli->query("INSERT INTO orders (id, address, customer\_name, status) VALUES ('$id', '$order->address', '$order->customer\_name', '$order->status') ON DUPLICATE KEY UPDATE address='$order->address', customer\_name='$order->customer\_name', status='$order->status'");

$mysqli->query("DELETE FROM ordered\_goods WHERE order\_id='$id'");

foreach ($order->goods as $good) {

$good\_id = $good['id'];

$quantity = $good['quantity'];

$mysqli->query("INSERT INTO ordered\_goods (order\_id, good\_id, quantity) VALUES ('$id', '$good\_id', '$quantity')");

}

return $id;

}

public static function partialUpdate(int $id, OrderCreate $order): ?int

{

global $mysqli;

if (isset($order->address))

$mysqli->query("UPDATE orders SET address='$order->address' WHERE id='$id'");

if (isset($order->customer\_name))

$mysqli->query("UPDATE orders SET customer\_name='$order->customer\_name' WHERE id='$id'");

if (isset($order->status))

$mysqli->query("UPDATE orders SET status='$order->status' WHERE id='$id'");

if (isset($order->goods)) {

$mysqli->query("DELETE FROM ordered\_goods WHERE order\_id='$id'");

foreach ($order->goods as $good) {

$good\_id = $good['id'];

$quantity = $good['quantity'];

$mysqli->query("INSERT INTO ordered\_goods (order\_id, good\_id, quantity) VALUES ('$id', '$good\_id', '$quantity')");

}

}

return $id;

}

public static function delete($id): bool

{

global $mysqli;

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

return $mysqli->affected\_rows > 0;

}

}

Были написаны контроллеры для обработки CRUD-операций:

Листинг 4 – goodAPI.php

<?php

require\_once 'api/api.php';

require\_once 'repository/goodRepository.php';

class GoodAPI extends API

{

public $apiName = 'goods';

public function indexAction()

{

$goods = GoodRepository::getAll();

if ($goods) {

return $this->response($goods, 200);

}

return $this->response('Data not found', 404);

}

public function viewAction()

{

$id = array\_shift($this->requestUri);

if ($id) {

$good = GoodRepository::getById($id);

if ($good)

return $this->response($good, 200);

}

return $this->response('Good not found', 404);

}

public function createAction()

{

$data = $this->requestJSON;

$good = new GoodCreate(

$data['title'],

$data['description'],

$data['price']

);

$good\_id = GoodRepository::create($good);

if ($good\_id)

return $this->response(GoodRepository::getById($good\_id), 201);

return $this->response('Good creation error', 500);

}

public function updateAction()

{

$id = array\_shift($this->requestUri);

$data = $this->requestJSON;

if (!$data || !isset($data['title']) || !isset($data['description']) || !isset($data['price'])) {

return $this->response('Bad request', 400);

}

$good = new GoodCreate(

$data['title'],

$data['description'],

$data['price']

);

$good\_id = GoodRepository::update($id, $good);

if ($good\_id)

return $this->response(GoodRepository::getById($good\_id), 200);

return $this->response('Good update error', 500);

}

public function partialUpdateAction()

{

$id = array\_shift($this->requestUri);

$data = $this->requestJSON;

if (!$data || !$id) {

return $this->response('Bad request', 400);

}

$good = GoodRepository::getById($id);

if (!$good) {

return $this->response('Good not found', 404);

}

$good = new GoodCreate(

$data['title'] ?? $good['title'],

$data['description'] ?? $good['description'],

$data['price'] ?? $good['price']

);

$good\_id = GoodRepository::update($id, $good);

if ($good\_id) {

return $this->response(GoodRepository::getById($good\_id), 200);

}

return $this->response('Good update error', 500);

}

public function deleteAction()

{

$id = array\_shift($this->requestUri);

if ($id) {

$deleted = GoodRepository::delete($id);

if ($deleted)

return $this->response('Good deleted', 204);

}

return $this->response('Good not found', 404);

}

}

Вывод списка всех товаров в Postman:



Рисунок 1. GET-запрос для вывода списка товаров

Листинг 5 – orderAPI.php

<?php

require\_once 'api/api.php';

require\_once 'repository/orderRepository.php';

class OrderAPI extends API

{

public $apiName = 'orders';

public function indexAction()

{

$orders = OrderRepository::getAll();

if ($orders) {

return $this->response($orders, 200);

}

return $this->response('Data not found', 404);

}

public function viewAction()

{

$id = array\_shift($this->requestUri);

if ($id) {

$order = OrderRepository::getById($id);

if ($order) {

return $this->response($order, 200);

}

}

return $this->response('Order not found', 404);

}

public function createAction()

{

$data = $this->requestJSON;

$order = new OrderCreate(

$data['address'],

$data['customer\_name'],

null,

$data['goods']

);

$order\_id = OrderRepository::create($order);

if ($order\_id) {

return $this->response(OrderRepository::getById($order\_id), 201);

}

return $this->response('Order creation error', 500);

}

public function updateAction()

{

$id = array\_shift($this->requestUri);

$data = $this->requestJSON;

if (!$data || !$id || !isset($data['address']) || !isset($data['customer\_name']) || !isset($data['goods'])) {

return $this->response('Bad request', 400);

}

$order = new OrderCreate(

$data['address'],

$data['customer\_name'],

$data['status'],

$data['goods'],

);

$order\_id = OrderRepository::update($id, $order);

if ($order\_id) {

return $this->response(OrderRepository::getById($order\_id), 200);

}

return $this->response('Order update error', 500);

}

public function partialUpdateAction()

{

$id = array\_shift($this->requestUri);

$data = $this->requestJSON;

if (!$data) {

return $this->response('Bad request', 400);

}

$order = new OrderCreate(

isset($data['address']) ? $data['address'] : null,

isset($data['customer\_name']) ? $data['customer\_name'] : null,

isset($data['status']) ? $data['status'] : null,

isset($data['goods']) ? $data['goods'] : null,

);

if (OrderRepository::getById($id) === null)

return $this->response('Order not found', 404);

$order\_id = OrderRepository::partialUpdate($id, $order);

if ($order\_id) {

return $this->response(OrderRepository::getById($order\_id), 200);

}

return $this->response('Order update error', 500);

}

public function deleteAction()

{

$id = array\_shift($this->requestUri);

if ($id) {

$deleted = OrderRepository::delete($id);

if ($deleted)

return $this->response('Order deleted', 204);

}

return $this->response('Order not found', 404);

}

}

Вывод списка всех заказов в Postman:

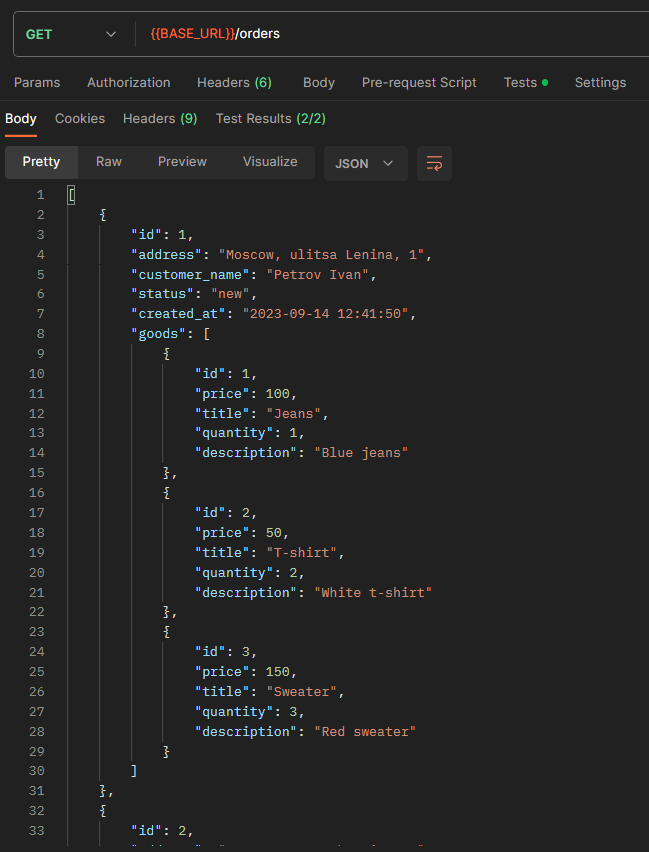


Рисунок 2. GET-запрос для вывода списка заказов

Для каждого контроллера, для каждого HTTP-метода были написаны тесты в Postman:

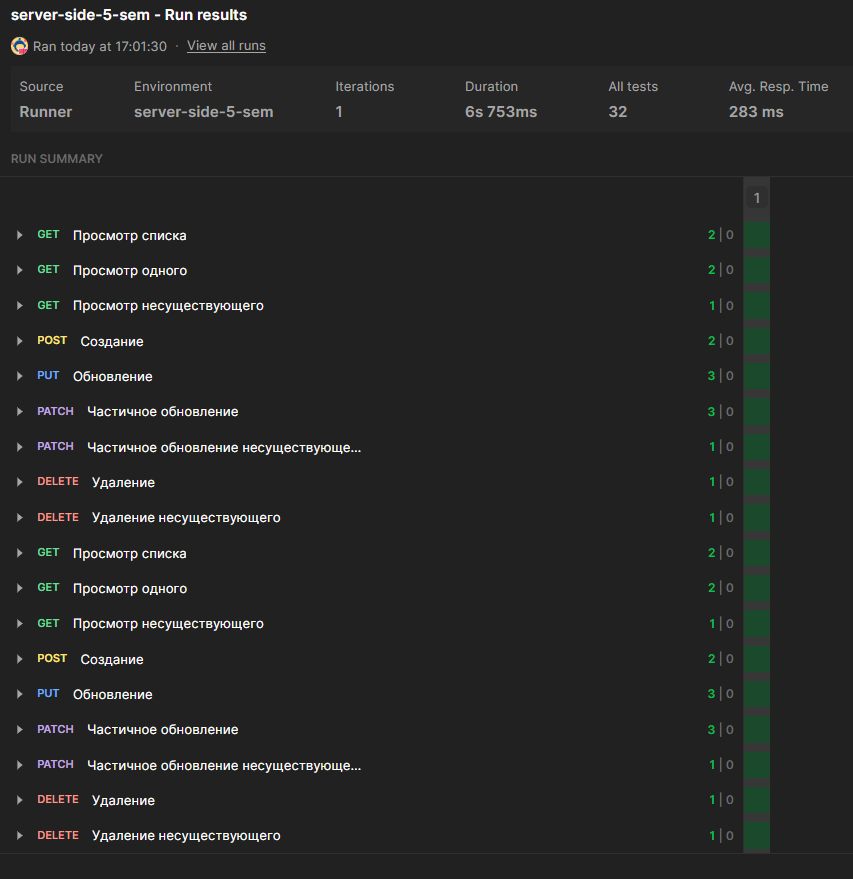


Рисунок 3. Успешные CRUD-тесты

**Вывод**

В ходе выполнения практической работы были получены навыки работы с базой данных и создания REST API.

**Ответы на вопросы**

1. **Что такое сущность?**

Сущность (entity) представляет тип объектов, которые должны храниться в базе данных. Каждая таблица в базе данных должна представлять одну сущность. Как правило, сущности соответствуют объектам из реального мира. У каждой сущности определяют набор атрибутов.

1. **Какие методы существуют в протоколе HTTP?**

Методы HTTP:

* 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.

1. **Что такое контроллер?**

Контроллер – это часть серверной логики, которая отвечает за обработку HTTP-запросов от клиента и управление ресурсами (например, базой данных или другими данными), связанными с этими запросами. Контроллеры в REST API обычно используются для выполнения операций CRUD (Create, Read, Update, Delete) над ресурсами, предоставляемыми API.

**Ссылка на удаленный репозиторий проекта**

<https://github.com/descenty/server-side-5-sem>

**Список использованной литературы**

1. Пример создания rest api на php: <https://cms.by/blog/php/primer-prostogo-sozdaniya-rest-api-na-php/>
2. Статья на хабре про принципы построения REST JSON API: <https://habr.com/ru/articles/447322/>
3. Лучшие практики rest api: https://habr.com/ru/companies/piter/articles/511382/