|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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** **№1 - 4**

по дисциплине «Разработка серверных частей интернет-ресурсов»

**Тема практической работы:**

**Студент группы** ИКБО-20-21 Сидоров Станислав

Дмитриевич

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

(подпись студента)

**Руководитель практической работы**  ассистент Благирев М.М.

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

(подпись руководителя)

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

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

Москва 2023

**Практическая работа №1**

**Цель работы:** создание конфигурации docker-compose для веб-сервера.

**Теоритическое введение:**

Backend – это серверная часть веб-приложения, которая обычно отвечает за обработку запросов пользователя, взаимодействие с базой данных, доступ к файлам и другим ресурсам. Она обеспечивает работу веб-приложения с общими ресурсами, включая базы данных, сервера и другие системы. Backend веб-приложения могут быть написаны на различных языках программирования, включая PHP, Java, Golang, Python и т.д.

Backend является одной из ключевых компонентов веб-приложения. Он обеспечивает логику, необходимую для обработки запросов, а также управление данными и их хранение. Backend веб-приложения работает в связке с frontend (клиентской частью), которая отображает пользовательский интерфейс и обрабатывает пользовательский ввод.

Таким образом, backend является неотъемлемой частью веб-приложения и обеспечивает его надежную работу и безопасность. Разработка backend части веб-приложения - это высоко востребованный навык в сфере IT.

На данном курсе вам необходимо будет выполнить 16 практических работ, которые разделены на 4 блока по языкам программирования. У первого блока практик язык программирования PHP. В ходе работы по данной практике вы должны будете создать первый простейший веб-сервер.

**Ход работы:**

Содержимое файла docker-compose.yml представлено в листинге 1.

Листинг 1 - содержимое файла docker-compose.yml

|  |
| --- |
| version: '3'  services:  db:  image: mysql  environment:  MYSQL\_USERNAME: user  MYSQL\_PASSWORD: password  MYSQL\_DATABASE: appDB  MYSQL\_ROOT\_PASSWORD: tmp  volumes:  - ./init.sql:/docker-entrypoint-initdb.d/init.sql  server:  build:  context: ./  dockerfile: dockerfile  ports:  - '8080:80'  volumes:  - ./:/var/www/html  links:  - db  volumes:  db\_data: |

Содержимое файла dockerfile представлено в листинге 2.

Листинг 2 - содержимое файла dockerfile

|  |
| --- |
| FROM php:7.4-apache  RUN docker-php-ext-install mysqli && docker-php-ext-enable mysqli  WORKDIR /var/www/html  COPY ./index.php /var/www/html  COPY ./style.css /var/www/html  EXPOSE 80 |

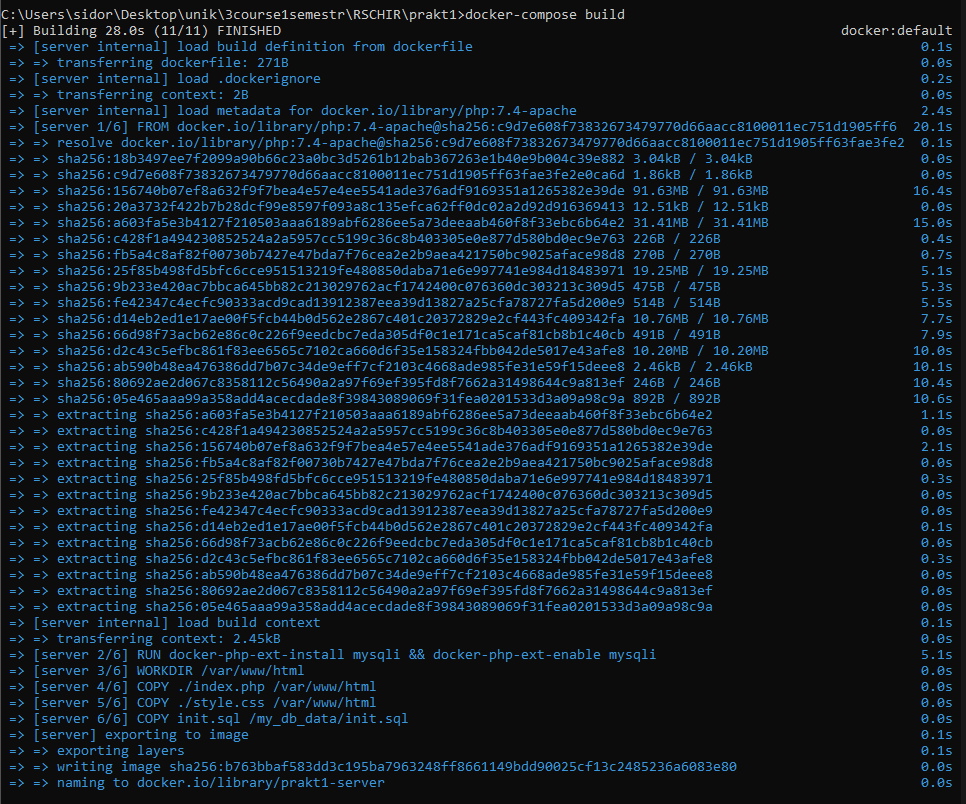
Результаты сборки, запуска и тестирования представлены на рисунках 1-3. 

Рисунок 1 - Успешная сборка

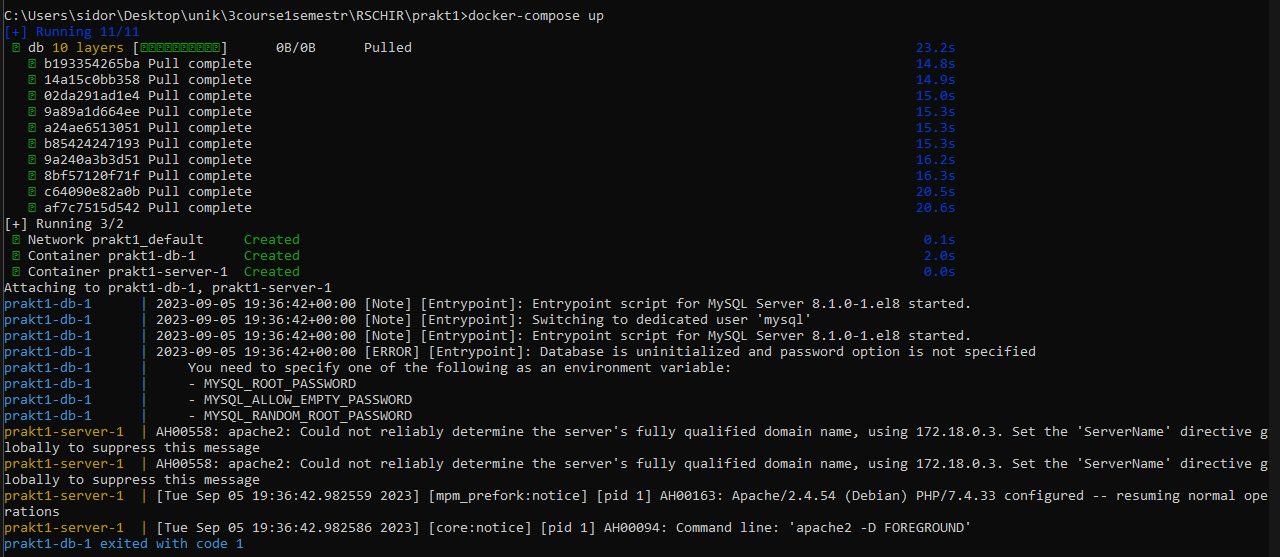


Рисунок 2 - Успешный запуск

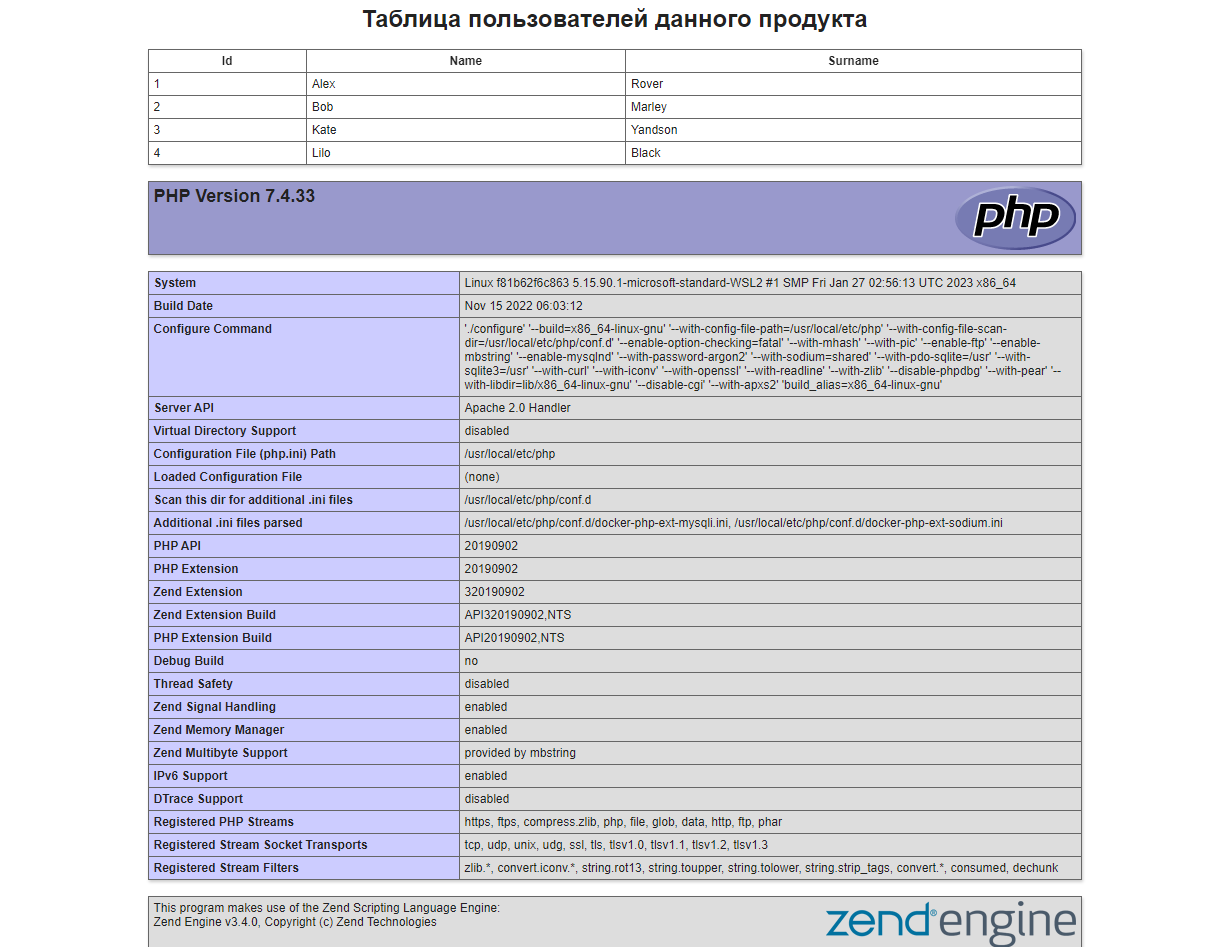


Рисунок 3 - Результат работы

**Вывод:**

В ходе практической работы были получены знания по использованию docker compose в развертке веб-сервера.

**Ответы на вопросы:**

1. Что такое apache и для чего он используется? Apache – открытое кросс-платформенное обеспечение, которое используется для размещения и поддержки веб-сервера.

2. Что такое оркестрация контейнеров и для чего это нужно? Оркестрация – программная технология, позволяющая автоматически управлять контейнерами, это нужно для разработки современных облачных приложений.

3. Что такое сервер? Сервер – устройство или программное обеспечение, способное принимать запросы и отвечать на них.

**Практическая работа №2**

**Задание:**

Вам необходимо создать конфигурацию docker-compose для вашего веб-сервера.

Docker контейнер веб сервера 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

Создать sql скрипт инициализации, где студенту необходимо реализовать любую одну сущность (например товары или студенты).

Реализовать связь между базой данных и веб-сервером.

Требуется создать 4 страницы php, где на каждой странице будет реализована одна из CRUD операций с выбранной вами сущностью.

**Теоритическое введени:**

CRUD - это акроним, который означает Create, Read, Update и Delete - четыре основных операции над данными в системах управления базами данных. Эти операции широко используются в веб-разработке для создания, чтения, обновления и удаления информации в базах данных.

**Ход выполнения:**

Измененное содержимое файла dockerfile представлено в листинге 3.

Листинг 3 - содержимое dockerfile

|  |
| --- |
| FROM php:7.4-apache  RUN docker-php-ext-install mysqli && docker-php-ext-enable mysqli  WORKDIR /var/www/html  COPY ./index.php /var/www/html  COPY ./create.php /var/www/html  COPY ./delete.php /var/www/html  COPY ./update.php /var/www/html  COPY ./style.css /var/www/html  EXPOSE 80 |

В ходе выполнения работы были созданы файлы delete.php, create.php, update.php, index.php их содержимое представлено в листингах 4 – 7 соответственно.

Листинг 4 - содержимое delete.php

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Deletion Page</title>  <link rel="stylesheet" href="style.css" type="text/css"/>  </head>  <body>  <?php  if ($\_SERVER['REQUEST\_METHOD'] == 'POST') {  $id = $\_POST['id'];    $query = "DELETE FROM users WHERE id = '$id'";    $mysqli = mysqli\_connect("db", "user", "password", "appDB");    if(mysqli\_query($mysqli, $query)){  header("Location: index.php");  } else{  echo "Ошибка: " . mysqli\_error($mysqli);  }  mysqli\_close($mysqli);  }  ?>  <h1>Deletion Page</h1>  <div class="form-container">  <form method="POST">  <h2>Deletion</h2>  <input type="number" name="id" placeholder="ID" required><br>  <button type="submit" name="delete">Delete</button>  </form>  </div>  <div class = "button-container">  <a href = "index.php">READ</a>  <a href = "update.php">UPDATE</a>  <a href = "create.php">CREATE</a>  </div>  </body>  </html> |

Листинг 5 - содержимое create.php

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Creation Page</title>  <link rel="stylesheet" href="style.css" type="text/css"/>  </head>  <body>  <?php  if ($\_SERVER['REQUEST\_METHOD'] == 'POST') {  $name = $\_POST['name'];  $surname = $\_POST['surname'];    $query = "INSERT INTO users (name, surname) VALUES ('$name', '$surname')";    $mysqli = mysqli\_connect("db", "user", "password", "appDB");  if(mysqli\_query($mysqli, $query)){  header("Location: index.php");  } else{  echo "Ошибка: " . mysqli\_error($mysqli);  }  mysqli\_close($mysqli);  }  ?>  <h1>Creation Page</h1>  <div class="form-container">  <form method="POST">  <h2>Creation</h2>  <input type="text" name="name" placeholder="Name" required><br>  <input type="text" name="surname" placeholder="Surname" required><br>  <button type="submit" name="create">Create</button>  </form>  </div>    <div class = "button-container">  <a href = "index.php">READ</a>  <a href = "delete.php">DELETE</a>  <a href = "update.php">UPDATE</a>  </div></body>  </html> |

Листинг 6 - содержимое update.php

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Update Page</title>  <link rel="stylesheet" href="style.css" type="text/css"/>  </head>  <body>  <?php  if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {  $id = $\_POST['id'];  $name = $\_POST['name'];  $surname = $\_POST['surname'];  $mysqli = mysqli\_connect("db", "user", "password", "appDB");  if ($id != null && $name != null){  $query = "UPDATE users SET name = '$name' WHERE id = $id";  if (!mysqli\_query($mysqli, $query )){  echo "Ошибка: " . mysqli\_error($mysqli);  echo "Request: " . $query;  }  }    if ($id != null && $surname != null){  $query = "UPDATE users SET surname = '$surname' WHERE id = $id";  if (!mysqli\_query($mysqli, $query)){  echo "Ошибка: " . mysqli\_error($mysqli);  echo "Request: " . $query;  }  }  mysqli\_close($mysqli);  }  ?>  <h1>Creation Page</h1>  <div class="form-container">  <form method="POST">  <h2>Creation</h2>  <input type="number" name="id" placeholder="ID" required><br>  <input type="text" name="name" placeholder="Name"><br>  <input type="text" name="surname" placeholder="Surname"><br>  <button type="submit" name="update">Update</button>  </form>  </div>  <div class = "button-container">  <a href = "index.php">READ</a>  <a href = "delete.php">DELETE</a>  <a href = "create.php">CREATE</a>  </div>  </body>  </html> |

Листинг 7 - содержимое index.php

|  |
| --- |
| <html lang="en">  <head>  <title>Read page</title>  <link rel="stylesheet" href="style.css" type="text/css"/>  </head>  <body>  <h1>Таблица пользователей данного продукта</h1>  <table>  <tr><th>Id</th><th>Name</th><th>Surname</th></tr>  <?php  $mysqli = mysqli\_connect("db", "user", "password", "appDB");  $result = mysqli\_query($mysqli, "SELECT \* FROM users");  foreach ($result as $row){  echo "<tr><td>{$row['ID']}</td><td>{$row['name']}</td><td>{$row['surname']}</td></tr>";  }  ?>  </table>    <div class = "button-container">  <a href = "update.php">UPDATE</a>  <a href = "delete.php">DELETE</a>  <a href = "create.php">CREATE</a>  </div>  </body>  </html> |

Результаты сборки, запуска и тестирования приведены на рисунках 4 – 9.

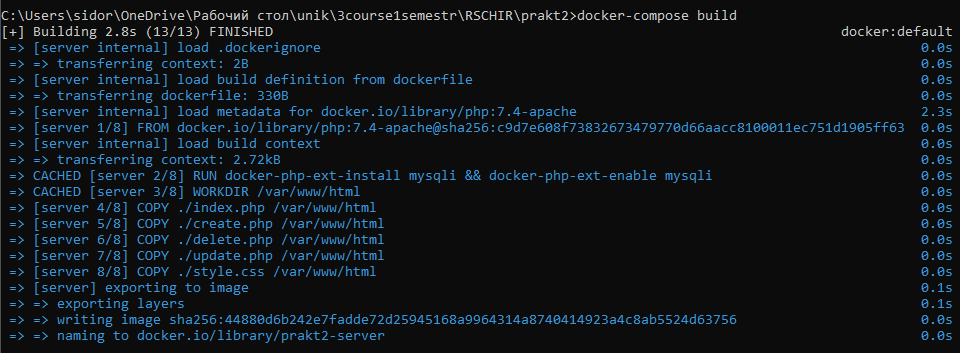


Рисунок 4 - результат сборки

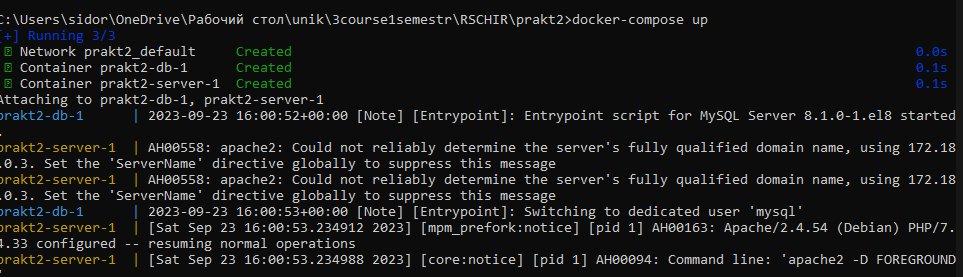


Рисунок 5 - результат запуска

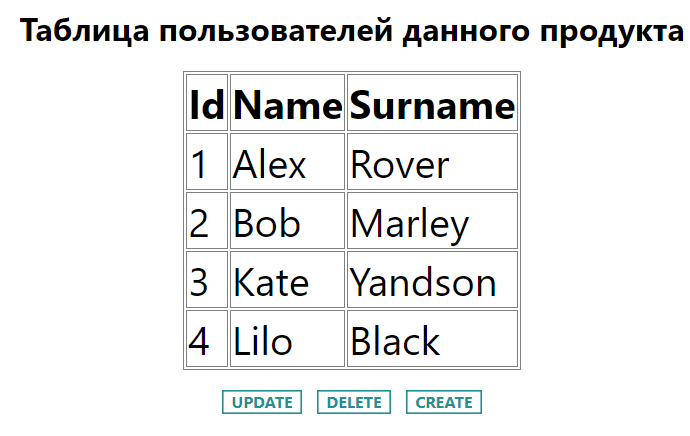


Рисунок 6 - страница index.php

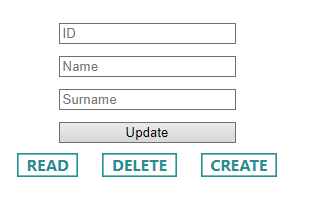


Рисунок 7 - страница update.php



Рисунок 8 - страница create.php

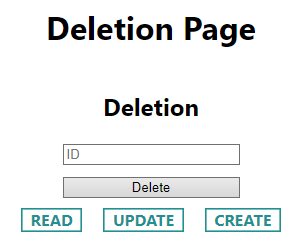


Рисунок 9 - страница delete.php

**Вывод:** в ходе данной практической работы были получены знания по работе с CRUD операциями через php.

**Ответы на вопросы:**

* Преимущества PHP для backend'а - простой синтаксис, большое сообщество, высокая производительность и большой выбор фреймворков и библиотек.
* Для достижения тонкого клиента при использовании PHP можно использовать AJAX для динамической загрузки данных с сервера.
* SSR - это техника веб-разработки, при которой контент страницы генерируется на сервере и доставляется клиенту в виде готовых HTML-страниц. В PHP SSR может быть реализован с помощью фреймворков, таких как Laravel или Symfony.

**Практическая работа №3**

**Задание:**

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

Docker контейнер веб серверва 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.

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

**Теоритическое введени:**

REST API (Representational State Transfer Application Programming Interface) - это архитектурный стиль, используемый в веб-разработке, который предоставляет доступ к приложению через интернет-протоколы, такие как HTTP и HTTPS. Учитывая, что web-приложения обычно состоят из клиентской и серверной частей, REST API находится в backend серверной части. REST API определяет набор ограничений и ограничения на взаимодействие между клиентом и сервером. Клиенты отправляют запросы с помощью определенных HTTP-методов (POST, GET, PUT, PATCH, DELETE), чтобы получить, обновить, создать или удалить определенную информацию или ресурс в приложении.

**Ход работы:**

В ходе работы были созданы две сущности user и comment. Comment содержит поле user\_id отвечающее за пользователя, которому пренадлежит комментарий. Содержимое файлов init.sql, user.php и comment.php представлено в листингах 8 – 10 соответсвенно.

Листинг 8 - содержимое файла init.sql

|  |
| --- |
| CREATE DATABASE IF NOT EXISTS appDB;  CREATE USER IF NOT EXISTS 'user'@'%' IDENTIFIED BY 'password';  GRANT SELECT,UPDATE,INSERT, DELETE ON appDB.\* TO 'user'@'%';  FLUSH PRIVILEGES;  USE appDB;  CREATE TABLE IF NOT EXISTS users (  id INT NOT NULL AUTO\_INCREMENT,  fullname VARCHAR(40) NOT NULL,  PRIMARY KEY (ID)  );  CREATE TABLE IF NOT EXISTS comments(  id INT AUTO\_INCREMENT PRIMARY KEY,  id\_user INT NOT NULL,  title VARCHAR(50),  content VARCHAR(256)  );  ALTER TABLE comments ADD FOREIGN KEY (id\_user) REFERENCES users(id); |

Листинг 9 - содержимое файла user.php

<?php

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

ini\_set('display\_errors', 1);

ini\_set('display\_startup\_errors', 1);

error\_reporting(E\_ALL);

$mysql = mysqli\_connect("db", "user", "password", "appDB");

$result = array();

if ($mysql){

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

parse\_str($\_SERVER['QUERY\_STRING'], $data);

if (!array\_key\_exists('id', $data)){

$resDB = mysqli\_query($mysql, 'SELECT \* FROM users');

if ($resDB != false){

$result['result'] = 'Success';

foreach($resDB as $row){

$result['items'][] = array("id" => $row['id'], "fullname" => $row['fullname']);

}

}

else{

$result['result'] = "Database error";

$result['message'] = mysqli\_error($mysql);

}

}else{

$resDB = mysqli\_query($mysql, "SELECT \* FROM users WHERE `id` = '" . $data['id'] . "'");

if ($resDB != false){

$result['result'] = 'Success';

foreach($resDB as $row){

$result['items'][] = array("id" => $row['id'], "fullname" => $row['fullname']);

}

}

else{

$result['result'] = "Database error";

$result['message'] = mysqli\_error($mysql);

}

}

}

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

$data = $\_POST;

if ($data['fullname'] != null){

$resDB = mysqli\_query($mysql, "INSERT INTO users(fullname) VALUES ('" . $data['fullname'] . "')");

$result['result'] = $resDB;

}else{

$result['result'] = "Request error";

$result['message'] = "Wrong passed data";

}

}

else if ($\_SERVER['REQUEST\_METHOD'] == 'PUT'){

parse\_str(file\_get\_contents("php://input"),$data);

if ($data['fullname'] != null && $data['id'] != null){

$resDB = mysqli\_query($mysql, "UPDATE users SET `fullname` = '" . $data['fullname'] . "' WHERE `id` = '" . $data['id'] . "'");

$result['result'] = $resDB;

}else{

$result['result'] = "Request error";

$result['message'] = "Wrong passed data";

}

}

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

parse\_str(file\_get\_contents("php://input"),$data);

if ($data['id'] != null){

$resDB = mysqli\_query($mysql, "DELETE FROM users WHERE `id` = '" . $data['id'] . "'");

$result['result'] = $resDB;

}else{

$result['result'] = "Request error";

$result['message'] = "Wrong passed data";

}

}

}

else{

$result['result'] = "Database error";

$result['message'] = mysqli\_error($mysql);

}

echo json\_encode($result);

Листинг 10 - содержимое файла comment.php

<?php

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

ini\_set('display\_errors', 1);

ini\_set('display\_startup\_errors', 1);

error\_reporting(E\_ALL);

$mysql = mysqli\_connect("db", "user", "password", "appDB");

$result = array();

if ($mysql){

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

parse\_str($\_SERVER['QUERY\_STRING'], $data);

if (!array\_key\_exists('id', $data)){

$resDB = mysqli\_query($mysql, 'SELECT \* FROM comments');

if ($resDB != false){

$result['result'] = 'Success';

foreach($resDB as $row){

$result['items'][] = array("id" => $row['id'], "title" => $row['title'], "content" => $row['content']);

}

}

else{

$result['result'] = "Database error";

$result['message'] = mysqli\_error($mysql);

}

}else{

$resDB = mysqli\_query($mysql, "SELECT \* FROM comments WHERE `id` = '" . $data['id'] . "'");

if ($resDB != false){

$result['result'] = 'Success';

foreach($resDB as $row){

$result['items'][] = array("id" => $row['id'], "title" => $row['title'], "content" => $row['content']);

}

}

else{

$result['result'] = "Database error";

$result['message'] = mysqli\_error($mysql);

}

}

}

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

$data = $\_POST;

if ($data['title'] != null && $data['content'] != null && $data['id\_user'] != null){

$resDB = mysqli\_query($mysql, "INSERT INTO comments(id\_user, title, content) VALUES ('" . $data['id\_user'] . "', '" . $data['title'] . "', '" . $data['content'] . "')");

$result['result'] = $resDB;

if ($resDB == false){

$result['error'] = mysqli\_error($mysql);

}

}else{

$result['result'] = "Request error";

$result['message'] = "Wrong passed data";

}

}

else if ($\_SERVER['REQUEST\_METHOD'] == 'PUT'){

parse\_str(file\_get\_contents("php://input"), $data);

if ($data['title'] != null && $data['content'] != null && $data['id\_user'] != null && $data['id'] != null ){

$resDB = mysqli\_query($mysql, "UPDATE comments SET `id\_user` = '" . $data['id\_user'] . "', `title` = '" . $data['title'] . "', `content` = '" . $data['content'] . "'WHERE `id` = '" . $data['id'] . "'");

$result['result'] = $resDB;

}else{

$result['result'] = "Request error";

$result['message'] = "Wrong passed data";

}

}

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

parse\_str(file\_get\_contents("php://input"),$data);

if ($data['id'] != null){

$resDB = mysqli\_query($mysql, "DELETE FROM comments WHERE `id` = '" . $data['id'] . "'");

$result['result'] = $resDB;

}else{

$result['result'] = "Request error";

$result['message'] = "Wrong passed data";

}

}

}

else{

$result['result'] = "Database error";

$result['message'] = mysqli\_error($mysql);

}

echo json\_encode($result);

Результаты сборки запуска и тестирования представлены на рисунках 9 – 11.

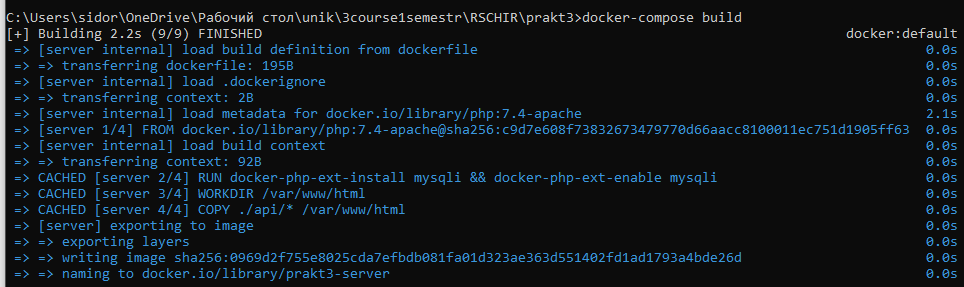


Рисунок 9 - Результат сборки

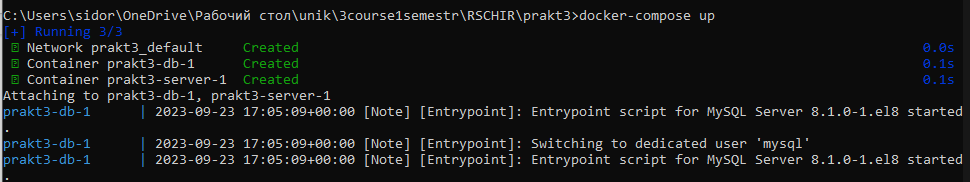


Рисунок 10 - Результат запуска

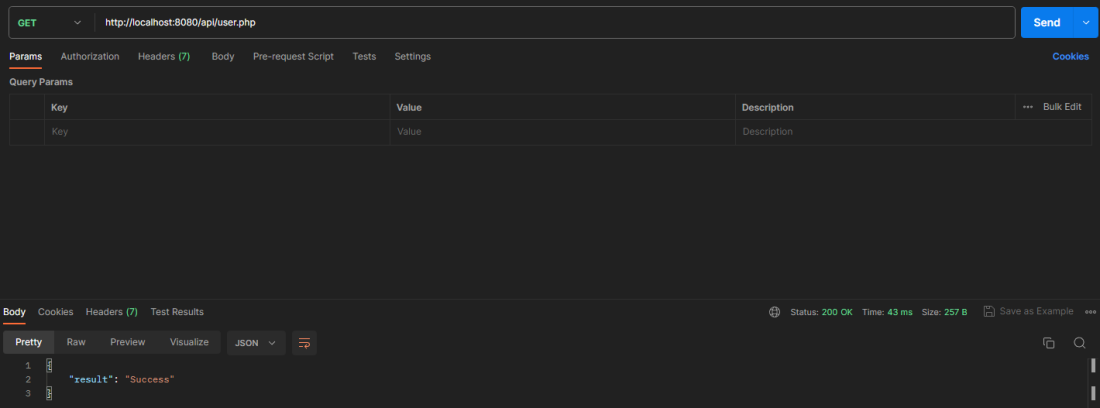


Рисунок 11 - Результат проверки

**Вывод:** в ходе выполнения данной практической работе были получены знания по созданию REST API с использованием php.

**Ответы на вопросы:**

* Сущность - это объект или концепция с определенными свойствами и методами, которые могут быть использованы для выполнения определенных действий в программном коде или базе данных.
* В протоколе HTTP существуют различные методы, такие как GET, POST, PUT, DELETE, HEAD и OPTIONS, которые могут быть использованы для отправки запросов к серверу.
* Контроллер - это компонент приложения, который управляет логикой обработки запросов и координирует работу моделей и видов в MVC-архитектуре. Он используется в языках программирования, таких как PHP, Java, Ruby и других.

**Практическая работа №4**

**Задание:**

Вам необходимо создать конфигурацию docker-compose для вашего веб-сервера.

Docker контейнер веб сервера должен включать:

* ОС Linux
* Веб-сервер Apache

Docker контейнер базы данных должен включать:

* ОС Linux
* Базу данных Mysql

Docker контейнер NGINX должен включать:

* ОС Linux
* NGINX

Docker-compose:

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

В рамках данного курса вам предлагается создать один большой проект маркетплейса, некоторые из практик, например, как эта, содержат кусок того проекта.

1. Создать 2 статичные html страницы для маркетплейса, где на одной странице будет информация по доставке, а на другой будут контакты и реквизиты вашего маркетплейса. (Эти страницы будут в финальном проекте)

2. Нужно сделать таким образом, чтобы статические страницы хранились в контейнере вместе с сервисом обратного прокси-сервера, а также настроить NGINX так, чтобы он мог их выдавать по запросу.

3. Создать 2 статичные html страницы. Динамические страницы могут быть взяты из предыдущих практических заданий или же написаны с нуля. Однако, необходимо сделать так, чтобы запросы и ответы на динамические страницы проходили через NGINX.

**Теоритическое введение:**

Nginx — это сервер веб-сервер и прокси-сервер, который обеспечивает высокую производительность и эффективную обработку запросов. Nginx изначально был создан для обеспечения высокой производительности в условиях высоких нагрузок на сервер, где он используется для обслуживания динамических веб-сайтов и приложений.

**Ход выполнения:**

В ходе выполнения данной работы были созданы файл конфигурации nginx, обновлен файл docker-compose.yml. В качетсве наполнения использовались страницы из предыдущих практических работ. Внутренне содержимое измененных файлов представлено в листингах 11 – 12.

Листинг 11 - содержимое файла nginx.conf

|  |
| --- |
| worker\_processes auto;    events {  worker\_connections 1024;  }    http {    server {    listen 80;    location ~ \.html$ {  root /usr/share/nginx/html;  }    location /favicon.ico {  deny all;  }    location ~ \.php$ {  proxy\_pass "<http://web-server>";  }    location / {  try\_files $uri $uri/ /index.php?$query\_string;  }  }  } |

Листинг 12 - содержимое файла docker-compose.yml

|  |
| --- |
| version: '3'    services:  db:  image: mysql:5.7  environment:  MYSQL\_USERNAME: user  MYSQL\_ROOT\_PASSWORD: password  MYSQL\_DATABASE: appDB  volumes:  - ./web-server/init.sql:/docker-entrypoint-initdb.d/init.sql    web-server:  build:  context: ./web-server  dockerfile: Dockerfile  ports:  - "8080:80"  volumes:  - ./web-server:/var/www/html  depends\_on:  - db    proxy:  build:  context: ./proxy  dockerfile: Dockerfile  ports:  - "80:80"  volumes:  - ./proxy/nginx.conf:/etc/nginx/nginx.conf  depends\_on:  - web-server |

Результат сборки и работы продемонстрирован на рисунках 12 – 13.

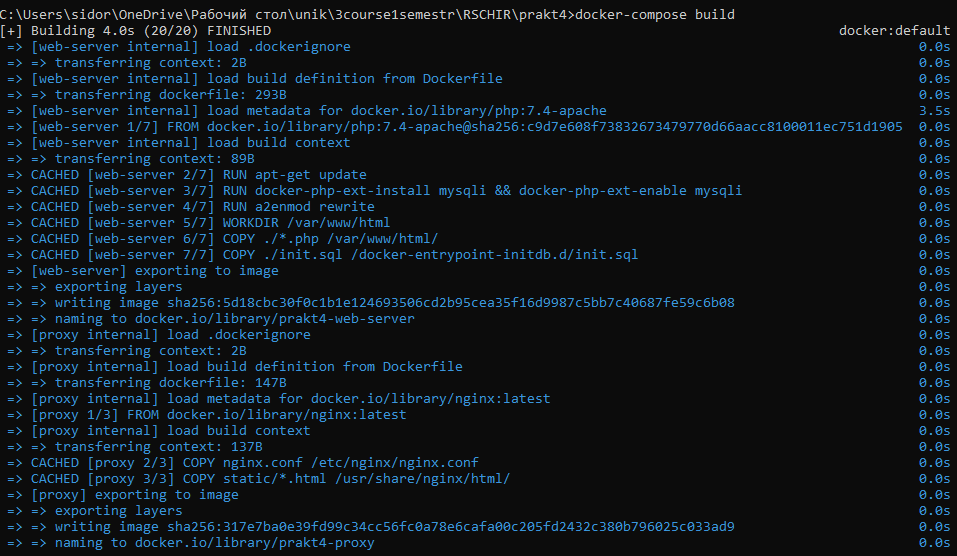


Рисунок 11 - Результат сборки

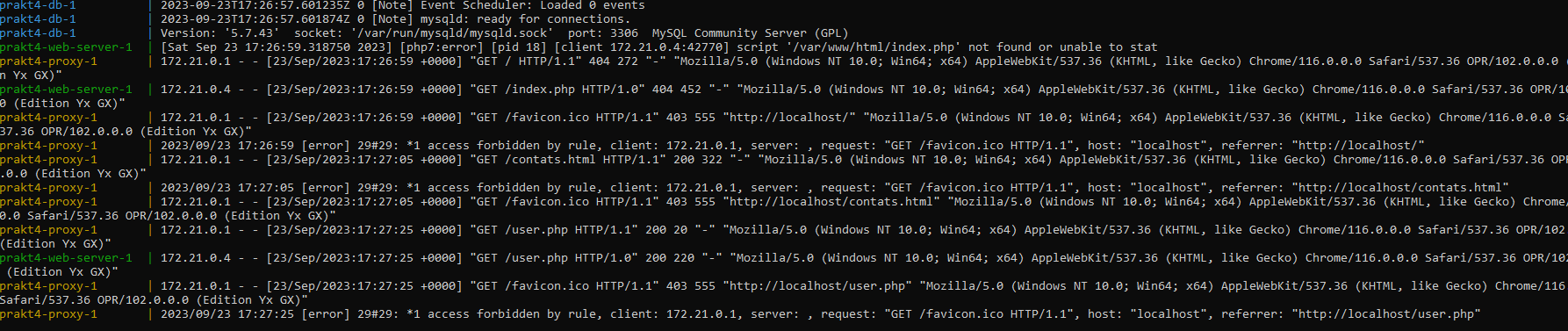


Рисунок 12 - Результат запуска и работы

**Выводы:** в результате данной практической работы были получены навыки по работе с nginx.

**Ответы на вопросы:**

* Обратный прокси сервер перенаправляет запросы от клиентов на веб-сервер.
* NGINX - высокопроизводительный веб-сервер и обратный прокси сервер, который может быть использован для обслуживания статических и динамических веб-сайтов, балансировки нагрузки, кеширования и других задач.
* Кеширование в NGINX позволяет хранить ответы от веб-сервера для быстрого доступа в будущем и может быть использовано для улучшения производительности и снижения нагрузки на сервер.
* Директива upstream в конфигурации NGINX позволяет добавлять несколько веб-серверов динамических страниц и определяет адреса серверов и методы балансировки нагрузки.

**Список использованных источников**

**⦁** Видео “Введение в Докер” на английском языке от создателя: Introduction to Docker (https://www.youtube.com/watch?v=Q5POuMHxW-0)

⦁ Статья о назначении докера простыми словами: https://habr.com/ru/post/309556/

⦁ Более сложная и подробная статья про докер: https://habr.com/ru/post/277699/

⦁ Хорошая статья с пингвинами для прочтения после туториала по докеру: https://habr.com/ru/post/250469/

⦁ Официальная документация докера: https://docs.docker.com/

⦁ Статья о конкретном опыте использования докер контейнеров: https://habr.com/ru/post/247969/

⦁ Туториал по докеру: https://badcode.ru/docker-tutorial-dlia-novichkov-rassmatrivaiem-docker-tak-iesli-by-on-byl-ighrovoi-pristavkoi/

⦁ Туториал по докеру с Хабра: https://habr.com/ru/post/310460/

⦁ Шпаргалка с командами Docker: https://habr.com/ru/company/flant/blog/336654/

⦁ Ссылка на скачивание докера с официального сайте: https://www.docker.com/products/docker-desktop

⦁ Отличная статья про dockerfile: https://habr.com/ru/company/ruvds/blog/439980/

⦁ Установка и настройка PHP: https://www.php.net/manual/ru/install.php

⦁ Настройка среды PhpStorm и полезные фичи: https://habr.com/ru/post/282003/

⦁ Про docker compose: https://habr.com/ru/company/ruvds/blog/450312/

⦁ Docker hub: https://hub.docker.com/