|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**ПРАКТИЧЕСКИЕ РАБОТЫ №1-4**

по дисциплине «Разработка серверных частей интернет-ресурсов»

**Студент группы** ИКБО-01-21 Кузнецов А.А.

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

(подпись студента)

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

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

(подпись руководителя)

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

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

Москва 2023

# ПРАКТИЧЕСКАЯ РАБОТА №1

# ЦЕЛЬ РАБОТЫ

Необходимо создать конфигурацию docker-compose для вашего веб-сервера.

Docker контейнер должен включать:

• ОС Linux

• Веб-сервер Apache

Docker-compose:

• Docker-контейнер, созданный ранее

• Примонтированные тома

• Настройки портов

Конечной задачей будет корректный запуск приложенного php скрипта генерации страницы с характеристиками веб-сервера и его работа на созданном веб-сервере.

# ХОД РАБОТЫ

Для создания образа необходимого веб-сервера, был использован Dockerfile, изображенные на рисунках 1-2.

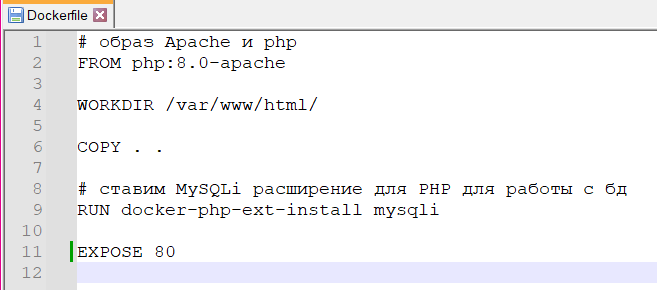


Рисунок 1 – Dockerfile для папки web

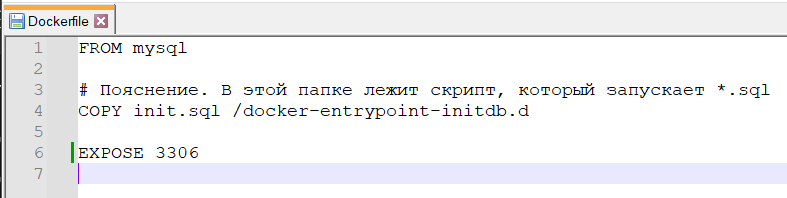


Рисунок 2 – Dockerfile для папки db

Здесь в качестве основы для нашего образа мы используем официальный образ PHP, затем копируем содержимое сервера, находящееся в текущей директории, в файловую систему веб-сервера и устанавливаем mysqli для корректной работы приложения.

Для связи сервера и его базы данных мы будем использовать docker-compose. Его содержимое показано на рисунке 3.

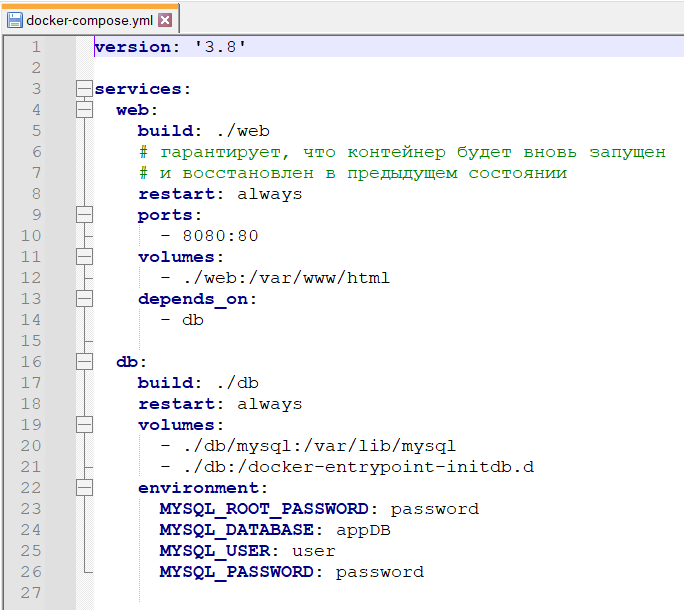


Рисунок 3 – docker-compose.yml

Здесь есть два сервиса: веб-сервер и база данных. При этом веб-сервер зависит от базы данных. В настройках сервера мы указываем Dockerfile, который создаст нужный нам образ, а также порты и тома для работы с нужными нам файлами в файловой системе сервера.

В сервисе базы данных мы, в свою очередь, задаем ее базовый образ и указываем переменные окружения: пользователя, название базы данных, корневой и пользовательский пароли. Помимо этого, мы указываем том для корректной инициализации базы данных на основе файла init.sql.

На рисунках 4-6 показана итоговая файловая структура проекта.

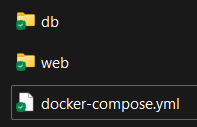


Рисунок 4 – Корневая папка



Рисунок 5 – Папка db для инициализации базы данных

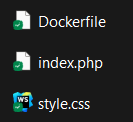


Рисунок 6 – Папка web для инициализации интерфейса веб-приложения

Проверим работу нашего сервера, выполнив команду “docker-compose up -d”, что показано на рисунке 7.

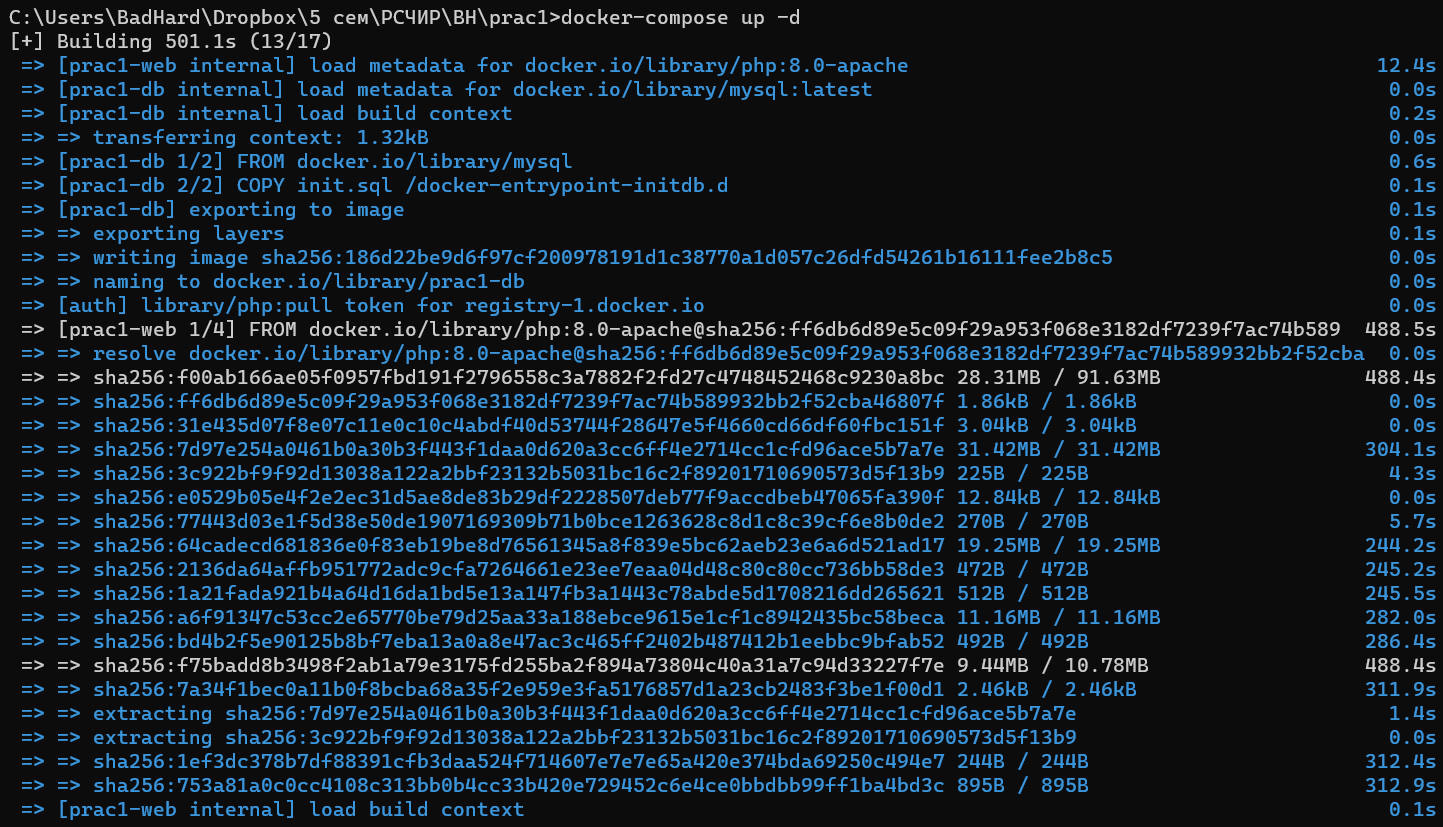


Рисунок 7 – Запуск сервера

Теперь можем перейти на localhost и убедиться, что все работает корректно, как показано на рисунке 8.

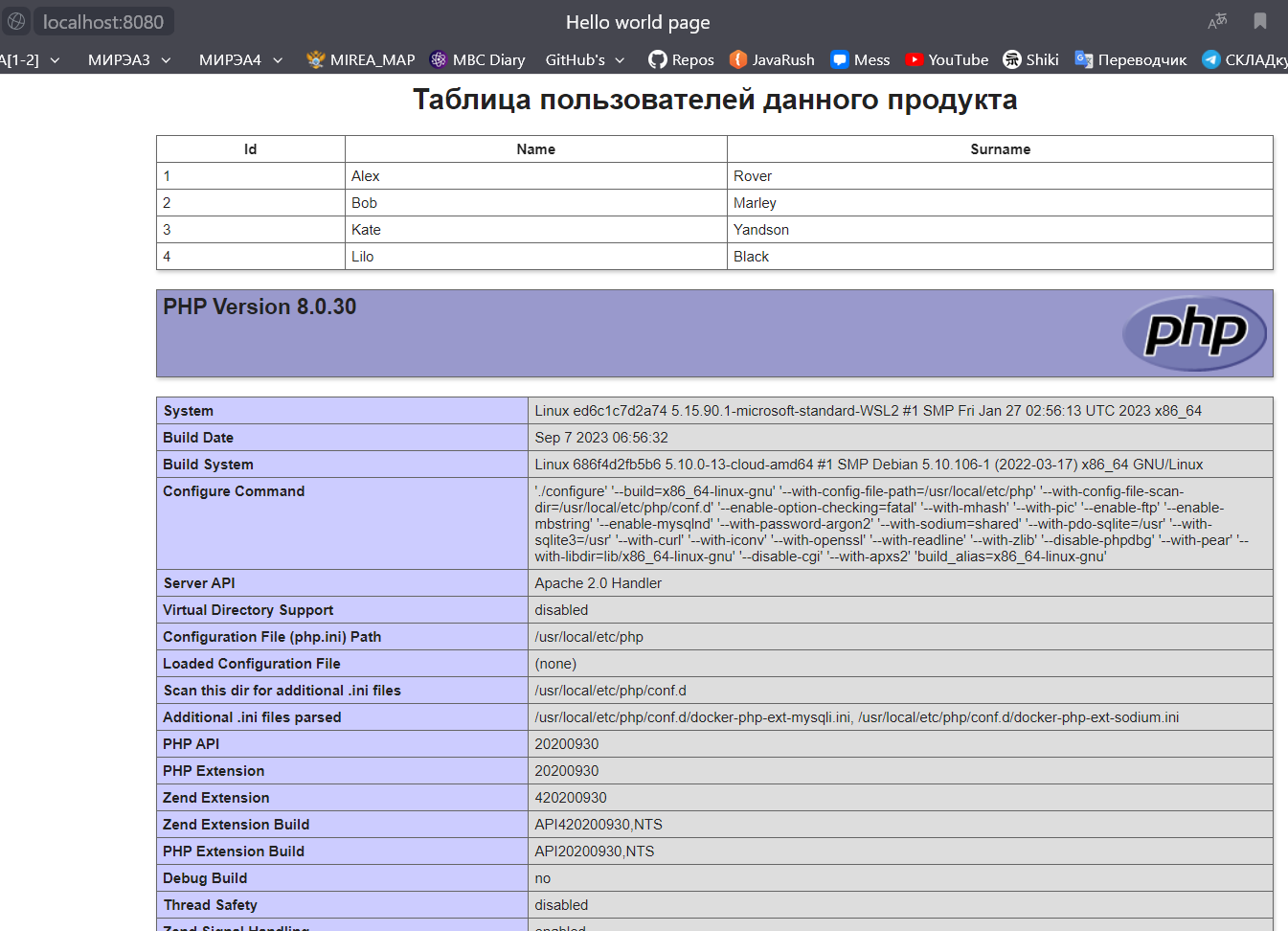


Рисунок 8 – Проверка работы сервера

# ВЫВОД

Таким образом, был произведен корректный запуск приложенного к практической работе php скрипта генерации страницы с характеристиками веб-сервера.

# ОТВЕТЫ НА ВОПРОСЫ

• Что такое Apache и для чего он нужен?

Apache — это веб-серверное программное обеспечение, используемое для хостинга и обслуживания веб-сайтов. Он нужен для обработки HTTP-запросов и доставки веб-страниц клиентам.

• Что такое оркестрация контейнеров и для чего это нужно?

Оркестрация контейнеров — это процесс управления и координации контейнеризированными приложениями, такими как Docker, в распределенной среде. Это нужно для автоматизации развертывания, масштабирования и управления контейнерами, обеспечивая их работоспособность и доступность.

• Что такое сервер?

Сервер — это вычислительное устройство или программное обеспечение, предназначенное для обработки запросов от клиентов и предоставления им необходимых ресурсов или услуг. Серверы могут использоваться для различных целей, включая хранение данных, предоставление сетевых услуг, хостинг веб-сайтов и многое другое.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Документация // PHP URL: <https://www.php.net/manual/ru/index.php> (дата обращения: 10.09.2023). – Текст: электронный.

2. Статья о назначении докера простыми словами: <https://habr.com/ru/post/309556/> (дата обращения: 10.09.2023).

3. Методические указания по выполнению практической работы: <https://online-edu.mirea.ru/mod/resource/view.php?id=508421> (дата обращения: 10.09.2023).

4. Официальная документация докера: <https://docs.docker.com/> (дата обращения: 10.09.2023).

5. Более сложная и подробная статья про докер: <https://habr.com/ru/post/277699/> (дата обращения: 10.09.2023).

# 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

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

2. Реализовать связь между базой данных и веб-сервером.

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

# ХОД РАБОТЫ

Для создания образа веб-сервера было использовано два Dockerfile. Первый был использован для образа веб-сервера, изображенный на рисунке 1, а также второй для инициализации БД с помощью скрипта, изображенный на рисунке 2.

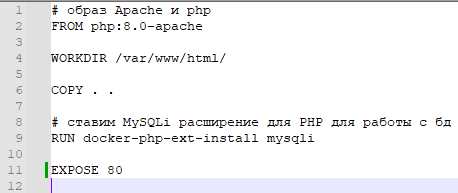


Рисунок 1 – Dockerfile для веб-сервера

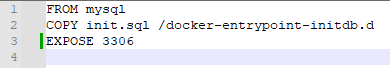


Рисунок 2 – Dockerfile для инициализации БД из файла

Затем мы создаем контейнер с помощью docker compose, чтобы связать образ базы данных с образом нашего веб-сервера, что показано на рисунке 3.



Рисунок 3 – docker-compose проекта

Для инициализации базы данных мы используем файл init.sql, показанный на рисунке 4.

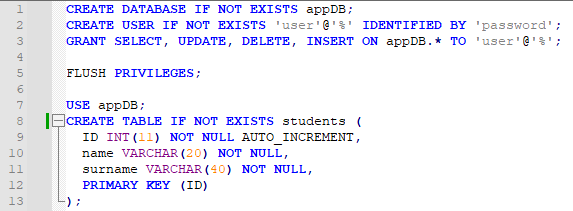


Рисунок 4 – Инициализация базы данных

Для реализации CRUD-команд были использованы php-скрипты. На рисунке 5 показан файл html-страницы со php-скриптом чтения записей read.php, а также его работа и проверка работоспособности в Postman изображена на рисунке 6.

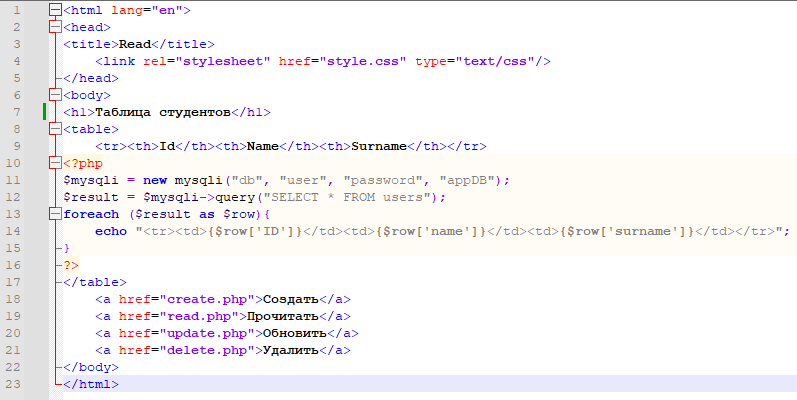


Рисунок 5 – Файл read.php

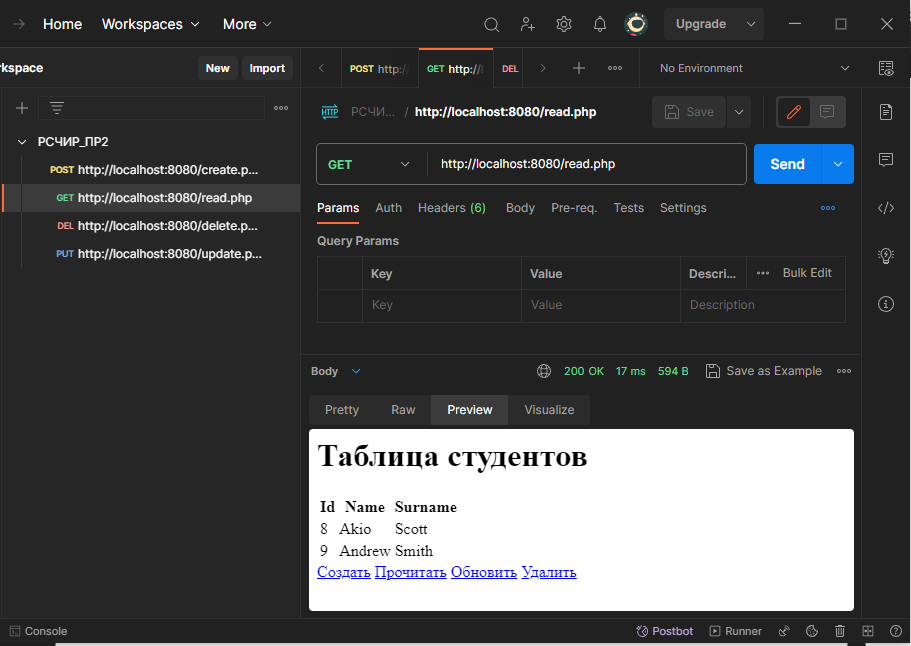


Рисунок 6 – Обработка запроса из Postman

На рисунке 7 показан файл html-страницы со php-скриптом добавления записей create.php, а также его работа и проверка работоспособности в Postman изображена на рисунке 8.



Рисунок 7 – Файл create.php

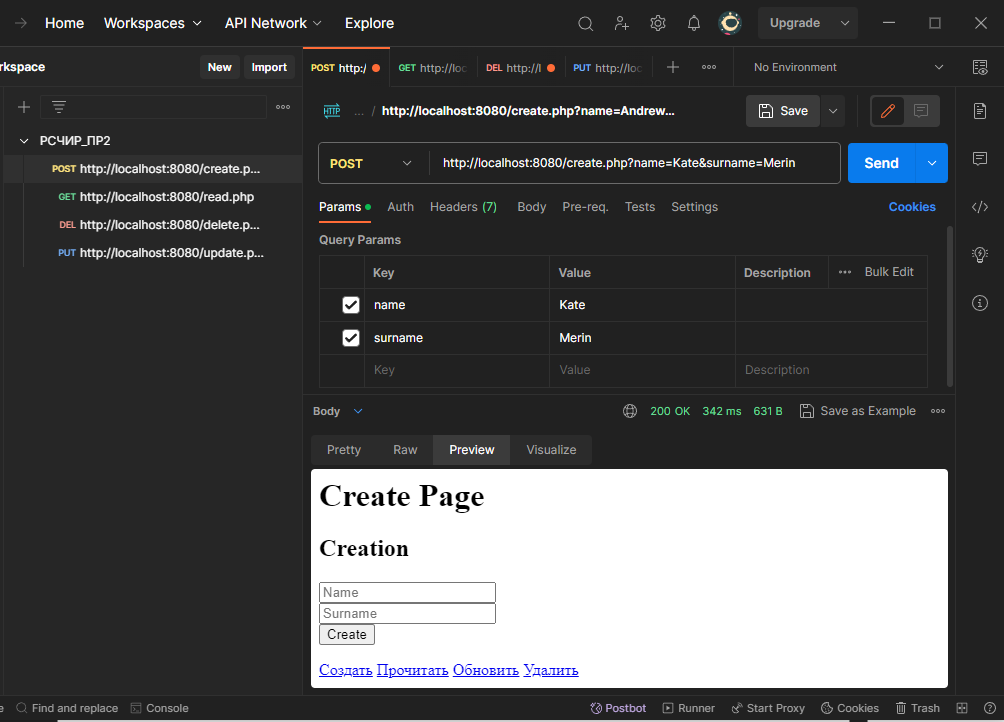


Рисунок 8 – Обработка запроса из Postman

На рисунке 9 показан файл html-страницы со php-скриптом изменения записей update.php, а также его работа и проверка работоспособности в Postman изображена на рисунке 10.

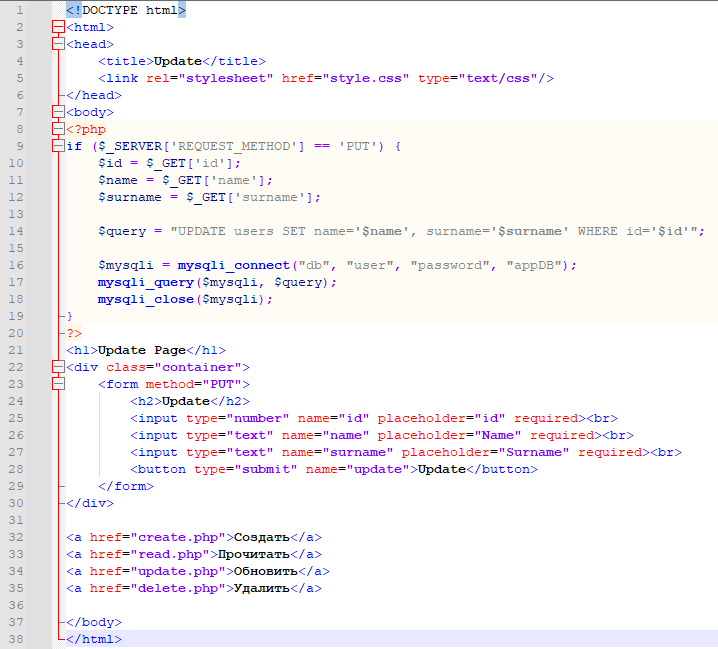


Рисунок 9 – Файл update.php

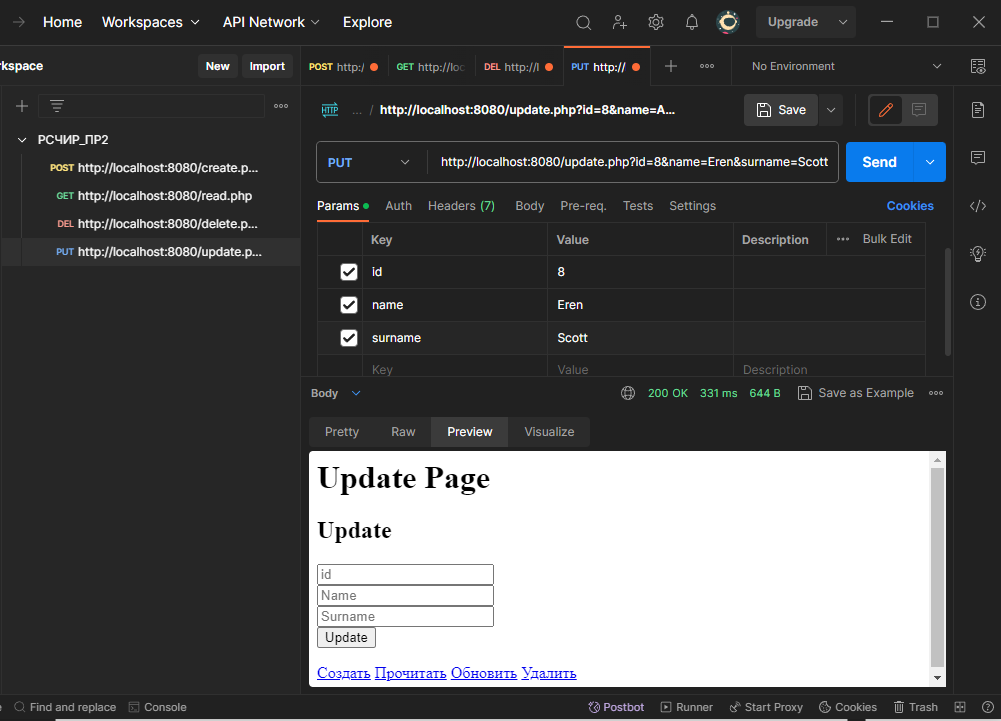


Рисунок 10 – Обработка запроса из Postman

На рисунке 11 показан файл html-страницы со php-скриптом удаления записей delete.php, а также его работа и проверка работоспособности в Postman изображена на рисунке 12.



Рисунок 11 – Файл delete.php

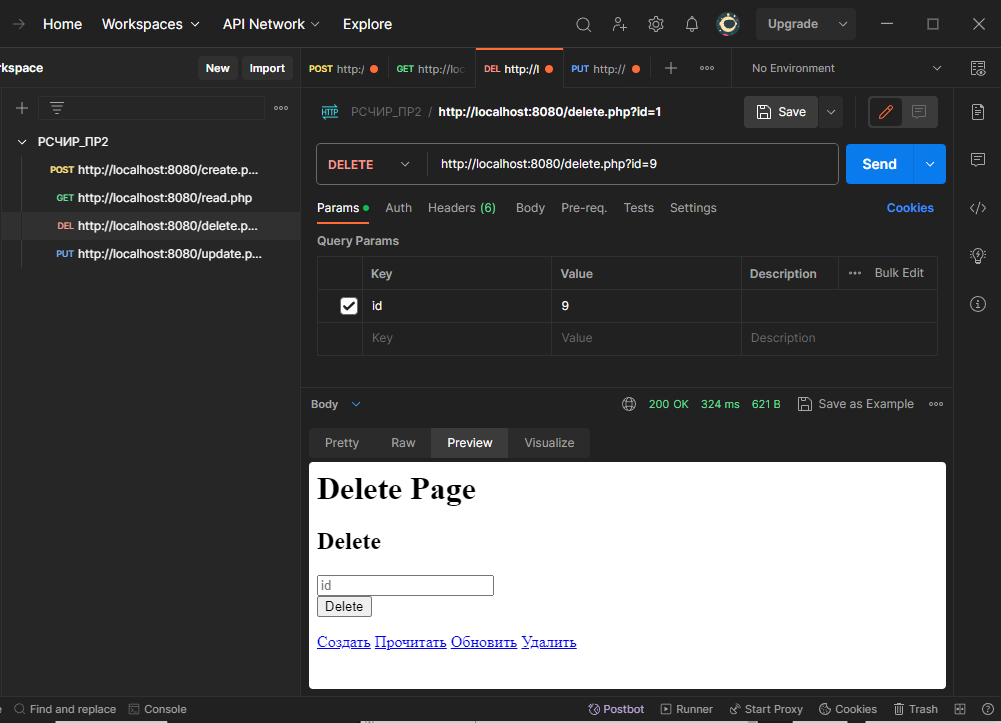


Рисунок 12 – Обработка запроса из Postman

Проверим результат работы ещё раз воспользовавшись запросом на получение записей, рисунок 13.

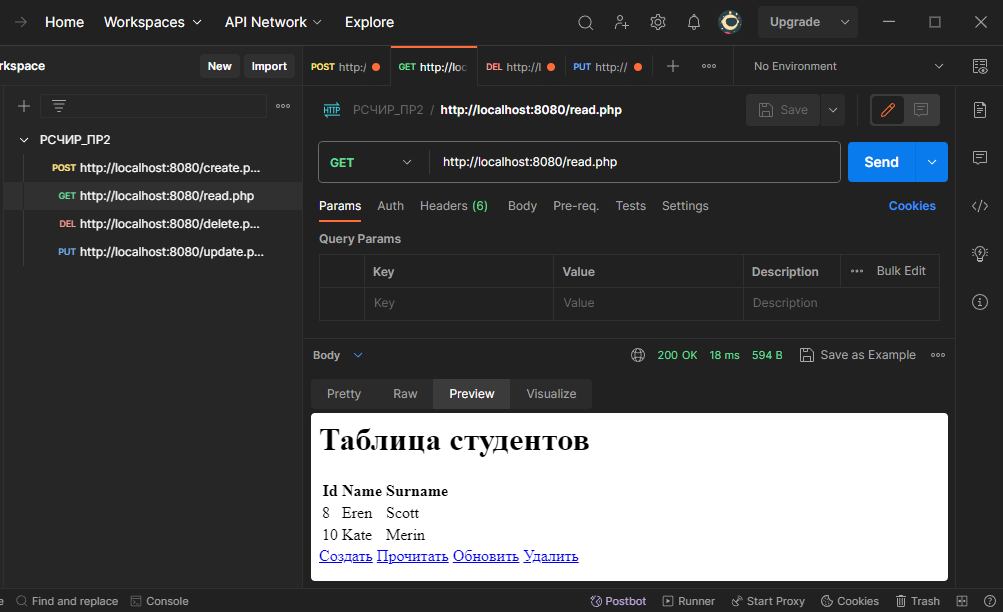


Рисунок 13 – Проверка результата выполнения запросов на сервер

# Вывод

Мы на практике убедились, что работа выполнена верно. В процессе работы было изучено создание конфигурации docker-compose для веб-сервера на базе Apache с использованием СУБД MySQL с примонтированным томом для БД и использованием php-скриптов на ряду с html-страницами с дальнейшей проверкой работы на них запросов в Postman.

# ОТВЕТЫ НА ВОПРОСЫ

1. В чём преимущество языка PHP для backend'а?

Преимущества PHP для backend'а включают:

* Простоту использования и изучения.
* Широкую поддержку баз данных и веб-серверов.
* Большое сообщество разработчиков и наличие множества библиотек и фреймворков.
* Встроенную поддержку веб-разработки, позволяющую вставлять PHP-код прямо в HTML-страницы.
* Возможность реализации как тонкого, так и толстого клиента.

1. Как достигается тонкий клиент при использовании PHP?

Тонкий клиент означает, что большая часть логики приложения выполняется на сервере. Для достижения этого с использованием PHP:

Логика приложения (контроллер) размещается на сервере, и сервер выполняет вычисления и обработку запросов.

Веб-сервер является "мозгом" приложения и отвечает за координацию работы сервера и клиентских устройств.

JavaScript на клиентской стороне используется для стандартных задач, таких как отображение данных и взаимодействие с пользователем.

Это позволяет минимизировать размер и сложность кода JavaScript, улучшить производительность и общее качество работы приложения.

1. Что такое SSR?

SSR (Server-Side Rendering) - это подход к созданию веб-страниц, при котором их начальный рендеринг происходит на сервере, а не на стороне клиента.

При использовании SSR веб-страницы генерируются на серверной стороне, и сервер отправляет готовый HTML-код браузеру для отображения пользователю.

SSR может быть использован с разными языками программирования, включая PHP, с использованием фреймворков и библиотек для рендеринга пользовательских интерфейсов.

Этот подход улучшает производительность и SEO-оптимизацию веб-страниц, так как страницы загружаются быстрее и более доступны для поисковых роботов.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Документация // PHP URL: <https://www.php.net/manual/ru/index.php> (дата обращения: 14.09.2023). – Текст: электронный.

2. Статья о назначении докера простыми словами: <https://habr.com/ru/post/309556/> (дата обращения: 14.09.2023).

3. Методические указания по выполнению практической работы: <https://online-edu.mirea.ru/mod/resource/view.php?id=508421> (дата обращения: 10.09.2023).

4. Официальная документация докера: <https://docs.docker.com/> (дата обращения: 14.09.2023).

5. Более сложная и подробная статья про докер: <https://habr.com/ru/post/277699/> (дата обращения: 14.09.2023).

# 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.

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

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

# ХОД РАБОТЫ

Файловая система работы изображена на рисунках 1-2.

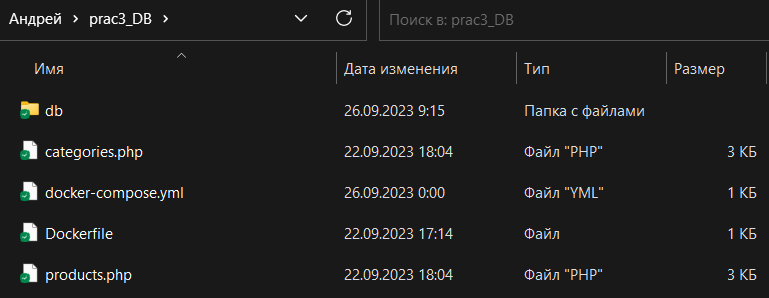


Рисунок 1 – Файловая система практической работы

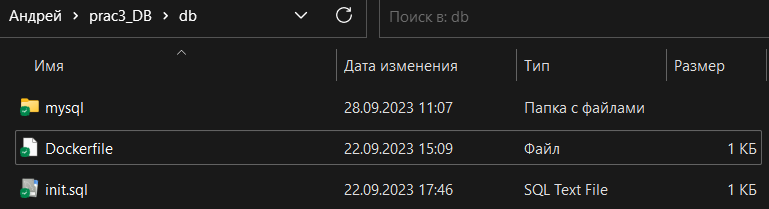


Рисунок 2 – Файловая система практической работы

На рисунках 3-4 изображены 2 Dockerfile с указанием образа PHP с поддержкой веб-сервера Apache с инструкцией, выполняющей команду внутри контейнера Docker, а также с установкой расширений mysqli, pdo и pdo\_mysql для Apache веб-сервера. Второй файл нужен для инициализации БД MySQL с помощью файла init.sql.

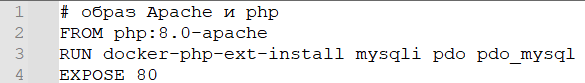


Рисунок 3 – Dockerfile для веб-сервера

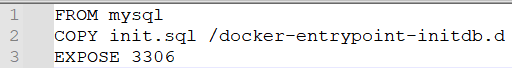


Рисунок 4 – Dockerfile для веб-сервера

Затем мы создаем контейнер с помощью docker compose, чтобы связать образ базы данных с образом нашего веб-сервера, что показано на рисунке 5.

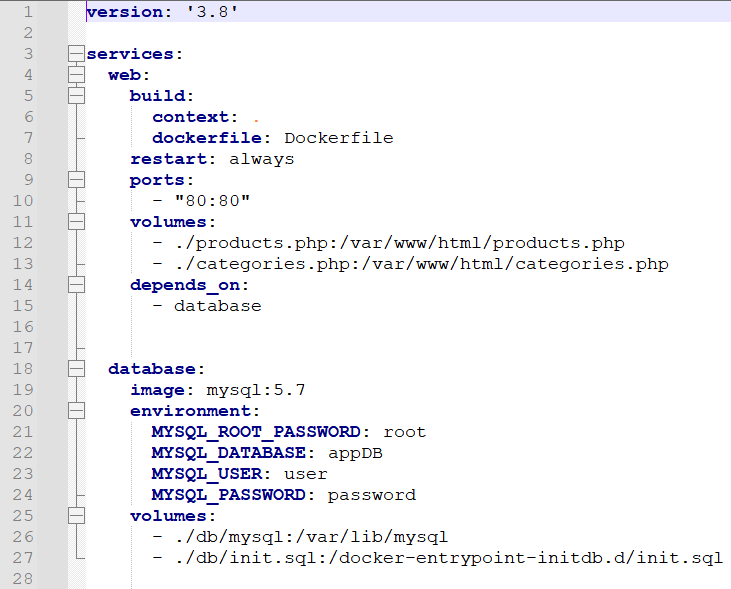


Рисунок 5 – docker-compose проекта

Для инициализации базы данных мы используем файл init.sql, показанный на рисунке 6.

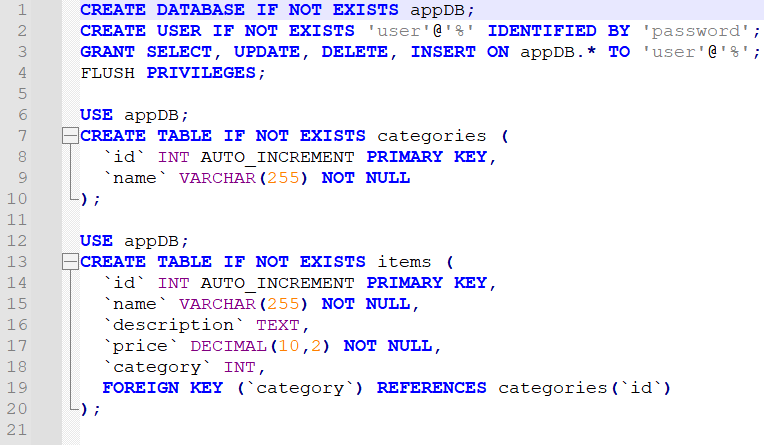


Рисунок 6 – Инициализация базы данных

Далее показан код categories.php на рисунках 7-11 показана реализация методов REST API.

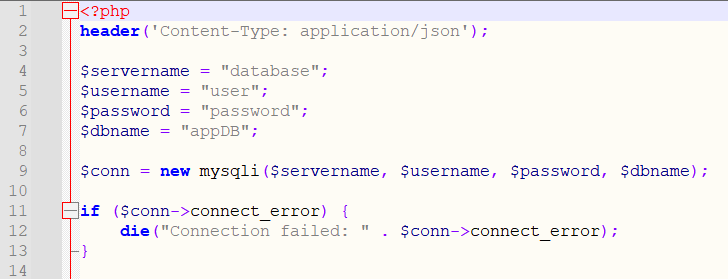


Рисунок 7 – Подключение скрипта к БД в categories.php

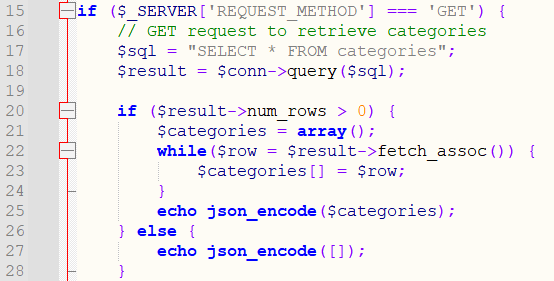


Рисунок 8 – Метод GET в categories.php

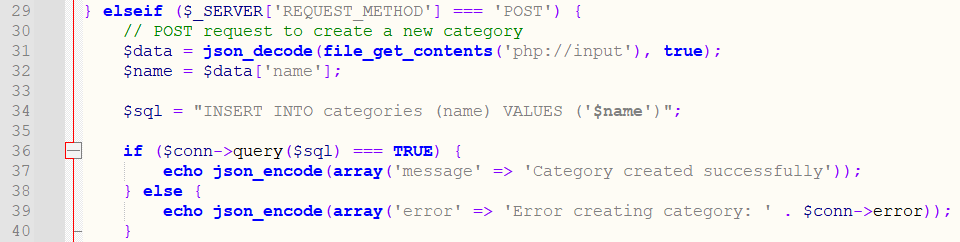


Рисунок 9 – Метод POST в categories.php

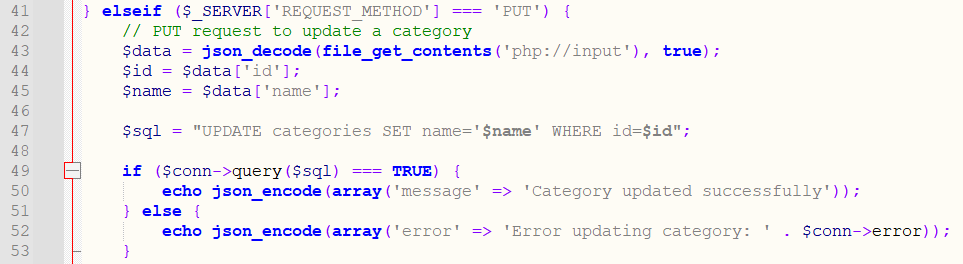


Рисунок 10 – Метод PUT в categories.php

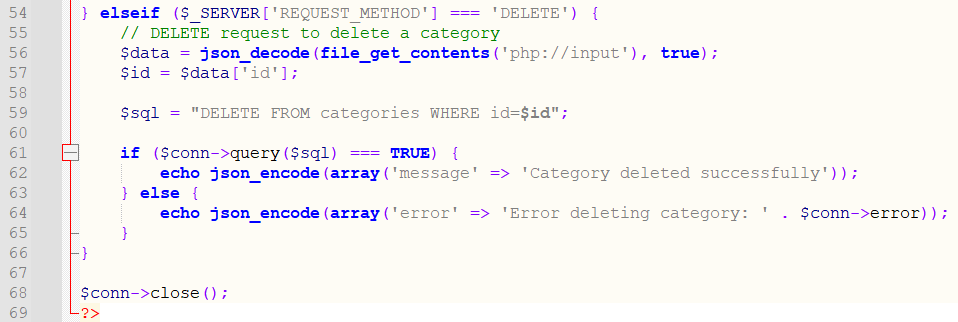


Рисунок 11 – Метод DELETE в categories.php

Далее показан код products.php на рисунках 12-16 показана реализация методов REST API.

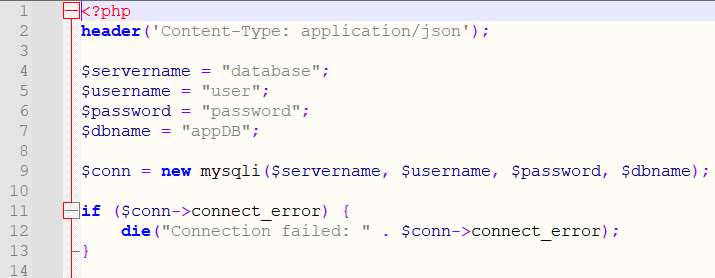


Рисунок 12 – Подключение скрипта к БД в products.php

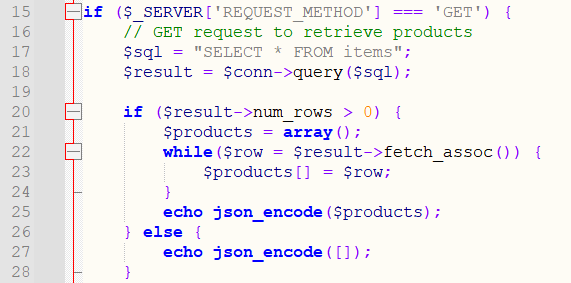


Рисунок 13 – Метод GET в products.php

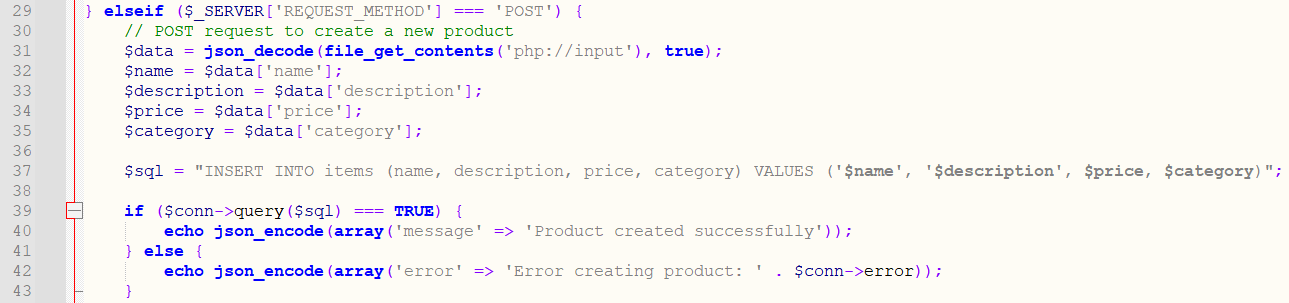


Рисунок 14 – Метод POST в products.php

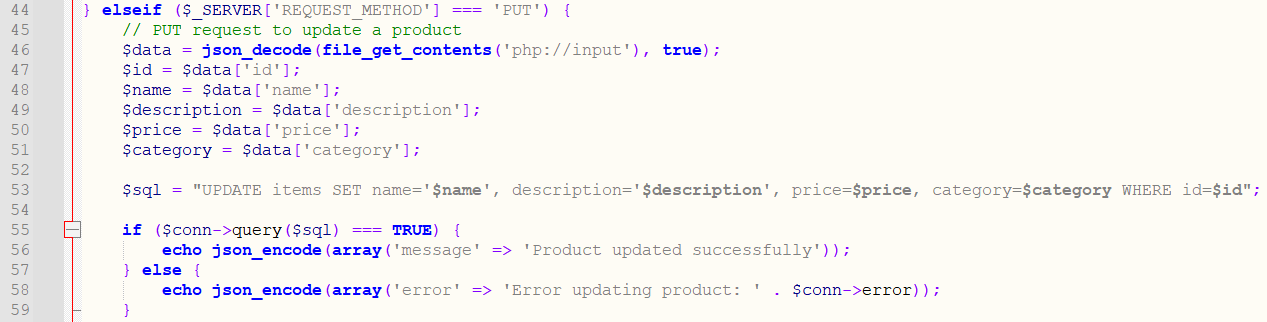


Рисунок 15 – Метод PUT в products.php

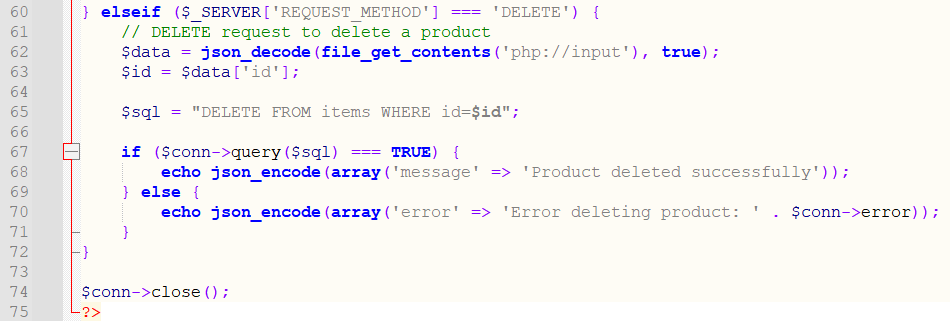


Рисунок 16 – Метод DELETE в products.php

# ОТВЕТЫ НА ВОПРОСЫ

1. Сущность:

В контексте информационных систем и баз данных, термин "сущность" обычно относится к объекту, о котором система хранит информацию. Сущность может быть чем-то конкретным, таким как человек, товар или заказ, или абстрактным, таким как концепция, представляющая собой некоторую информацию. Сущность обычно имеет атрибуты (свойства), которые описывают ее характеристики или параметры.

2. Методы в протоколе HTTP:

Протокол HTTP (Hypertext Transfer Protocol) определяет различные методы (или HTTP-методы), которые клиенты могут использовать для взаимодействия с веб-серверами. Некоторые из наиболее распространенных HTTP-методов включают:

- GET: Запрос на получение данных с сервера. Обычно используется для получения веб-страниц и ресурсов.

- POST: Отправка данных на сервер для обработки. Обычно используется для отправки форм и создания данных.

- PUT: Запрос на обновление данных на сервере. Обычно используется для обновления существующих ресурсов.

- DELETE: Запрос на удаление данных с сервера.

- HEAD: Аналогичен методу GET, но сервер возвращает только заголовки без тела ответа.

- PATCH: Запрос на частичное обновление данных.

- OPTIONS: Запрос на получение информации о возможных методах и параметрах, поддерживаемых сервером.

- TRACE: Запрос на диагностику пути, который прошел запрос к серверу.

3. Контроллер:

В контексте программирования и разработки веб-приложений, "контроллер" - это часть архитектуры MVC (Model-View-Controller), которая отвечает за обработку запросов от пользователей, управление бизнес-логикой и взаимодействие с моделью данных. Контроллер получает запросы от клиентов, определяет, какие действия выполнять, и взаимодействует с моделью (данными) и видом (пользовательским интерфейсом) для предоставления ответа.

Контроллеры являются ключевой частью многих веб-фреймворков и приложений, таких как Ruby on Rails, Django, ASP.NET MVC и другие. Они обеспечивают разделение логики приложения на компоненты, что делает код более организованным и легко поддерживаемым.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Документация // PHP URL: <https://www.php.net/manual/ru/index.php> (дата обращения: 16.10.2021). – Текст: электронный.

2. Дергачев, А. М. Проблемы эффективного использования сетевых сервисов / А. М. Дергачев // Научно-технический вестник СПбГУ ИТМО. –2011. – № 1 (71). С. 83-87.

3. Розенфельд, Л. Информационная архитектура в Интернете / Л. Розенфельд, П. Морвиль, 2 е издание. – Пер. с англ. – СПб: Символ Плюс, 2005 – 544 с.

4. Спинеллис, Д. Идеальная архитектура. Ведущие специалисты о красоте программных архитектур / Д. Спинеллис, Г. Гусиос. – Пер. с англ. –СПб.: Символ Плюс, 2010 – 528 с.

5. Фаулер, М. Архитектура корпоративных программных приложений / М. Фаулер Пер. с англ. − М.: Издательский дом "Вильяме", 2006 − 544 с.

6. Информационные технологии / О.Л. Голицына, Н.В. Максимов, Т.Л. Партыка, И.И. Попов. − 2-е изд. − Москва: ФОРУМ − ИНФРА-М, 2008. – 395 с.

7. Таненбаум, Э. Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. ван Стеен. − СПб.: Питер, 2003. − 93 с.

8. Котеров, Д.В. PHP 7 / Д.В. Котеров, И. В. Симдянов. - СПб.: БХВПетербург, 2021. – 1088 с.

# 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

# ХОД РАБОТЫ

Для создания образа веб-сервера, был использован Dockerfile, изображенный на рисунке 1.

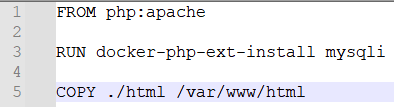


Рисунок 1 – Dockerfile для веб-сервера

Для связи базы данных, apache и nginx был использован файл docker compose, показанный на рисунке 2.

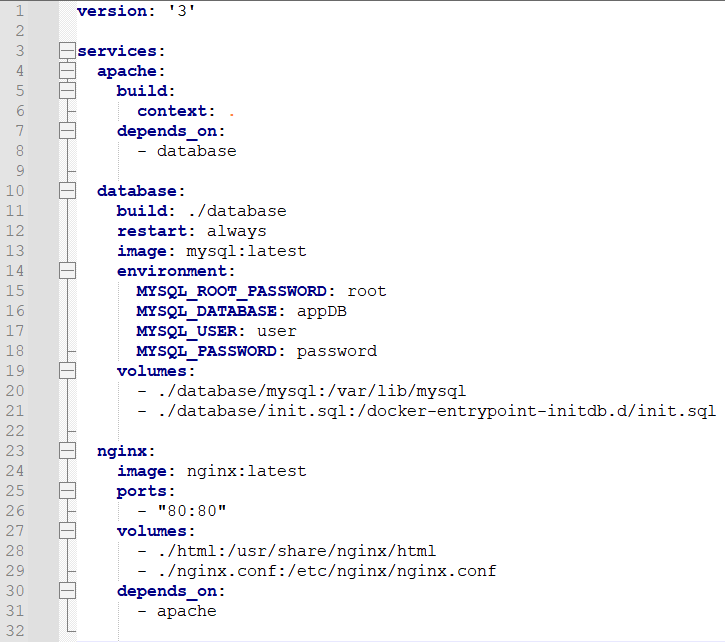


Рисунок 2 – docker-compose.yml

Также была создан файл nginx.conf для настройки конфигурации nginx-сервиса, что показано на рисунке 3.

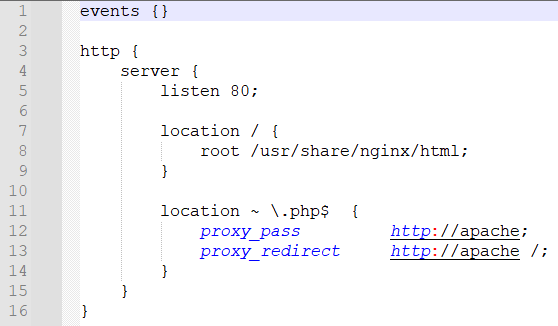


Рисунок 3 – nginx.conf

Работа сервисов внешне аналогична работе сервисов из предыдущей практической работы. Файловая система изображена на рисунках 4-6.

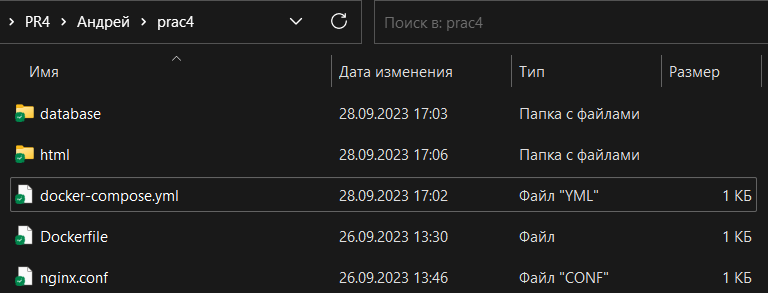


Рисунок 4 – Корневая папка проекта

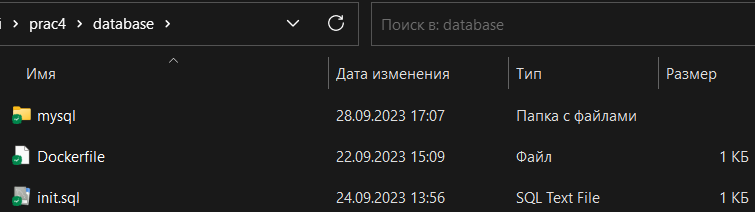


Рисунок 5 – Папка проекта для хранения записей в БД

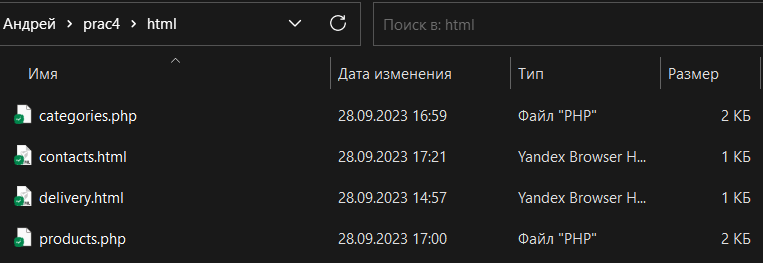


Рисунок 6 – Папка проекта с html-страницами и php-скриптами

# ОТВЕТЫ НА ВОПРОСЫ

1. Обратный прокси-сервер (Reverse Proxy Server):

Обратный прокси-сервер — это сервер, который принимает запросы от клиентов (обычно браузеров) и перенаправляет их к одному или нескольким целевым серверам (например, веб-серверам или приложениям). Обратный прокси-сервер выполняет функции, такие как балансировка нагрузки, кэширование, SSL терминация и защита от DDoS-атак, что позволяет улучшить производительность, надежность и безопасность веб-сервисов.

1. Использование NGINX:

NGINX — это веб-сервер и обратный прокси-сервер с открытым исходным кодом. Его можно использовать для следующих целей:

* Обслуживания статических и динамических веб-страниц.
* Проксирования запросов к приложениям и веб-серверам.
* Балансировки нагрузки между несколькими серверами.
* Кэширования содержимого для ускорения запросов.
* Терминации SSL и обеспечения безопасной связи.

1. Кеширование в NGINX:

NGINX поддерживает кеширование, что позволяет временно хранить статические ресурсы (например, HTML, CSS, JavaScript, изображения) на сервере для ускорения последующих запросов от клиентов. Когда клиент делает запрос к ресурсу, NGINX проверяет, есть ли этот ресурс в кеше. Если ресурс найден, он возвращается клиенту, и это уменьшает нагрузку на сервер.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Документация // PHP URL: <https://www.php.net/manual/ru/index.php> (дата обращения: 16.10.2021). – Текст: электронный.

2. Дергачев, А. М. Проблемы эффективного использования сетевых сервисов / А. М. Дергачев // Научно-технический вестник СПбГУ ИТМО. –2011. – № 1 (71). С. 83-87.

3. Розенфельд, Л. Информационная архитектура в Интернете / Л. Розенфельд, П. Морвиль, 2 е издание. – Пер. с англ. – СПб: Символ Плюс, 2005 – 544 с.

4. Спинеллис, Д. Идеальная архитектура. Ведущие специалисты о красоте программных архитектур / Д. Спинеллис, Г. Гусиос. – Пер. с англ. –СПб.: Символ Плюс, 2010 – 528 с.

5. Фаулер, М. Архитектура корпоративных программных приложений / М. Фаулер Пер. с англ. − М.: Издательский дом "Вильяме", 2006 − 544 с.

6. Информационные технологии / О.Л. Голицына, Н.В. Максимов, Т.Л. Партыка, И.И. Попов. − 2-е изд. − Москва: ФОРУМ − ИНФРА-М, 2008. – 395 с.

7. Таненбаум, Э. Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. ван Стеен. − СПб.: Питер, 2003. − 93 с.

8. Котеров, Д.В. PHP 7 / Д.В. Котеров, И. В. Симдянов. - СПб.: БХВПетербург, 2021. – 1088 с.