|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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**

по дисциплине «Разработка серверных частей интернет-ресурсов»

**Тема практической работы:**

**Студент группы** ИКБО-30-20 Патина Валерия Сергеевна

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

(подпись студента)

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

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

(подпись руководителя)

Работа представлена «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Москва 2022

**СОДЕРЖАНИЕ**

[Постановка задачи 1](#_Toc113565382)

[Выполнение работы 1](#_Toc113565383)

[Ответы на вопросы к практической: 4](#_Toc113565384)

[ВЫВОДЫ 9](#_Toc113565385)

Постановка задачи

Cоздать свою конфигурацию серверного программного обеспечения, в которой должны присутствовать веб-сервер, операционная система, язык программирования и база данных.

Для проверки работоспособности вашей конфигурации требуется инициализировать базу данных: создать отдельного пользователя для работы с ней, создать базу данных, в которой создать таблицу пользователи с полями: идентификационный номер, имя, фамилия. Также для проверки вашей конфигурации требуется сгенерировать тестовую страничку, содержащую выборку из созданной таблицы и информационное сообщение о версии языка программирования, его настройках и конфигурации.

Выполнение работы

Создадим рабочую директорию со структурой папок (рис.1)

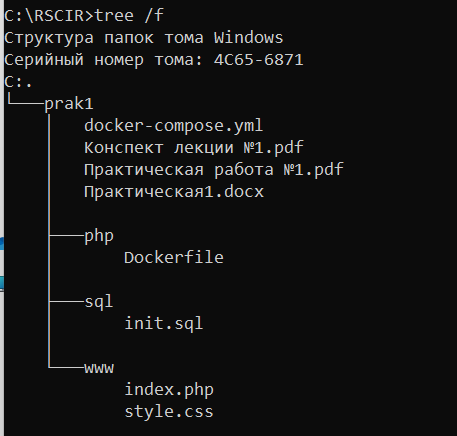


Рисунок 1 - Структура папок и файлов

Напишем Dockerfile (Листинг 1).

Листинг 1 – Dockerfile

FROM php:7.4-apache

RUN apt-get update && docker-php-ext-install mysqli

Исправим стандартный файл index.php. В итоге содержимое файла будет выглядеть следующим образом (Листинг 2).

Листинг 2 – index.php

<html lang=”en”>

<head>

<title>Hello world 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 = new mysqli(‘172.21.0.1’, ‘user’, ‘password’, ‘appDB’);

$result = $mysqli->query(“SELECT \* FROM users”);

foreach ($result as $row){

    echo “<tr><td>{$row[‘ID’]}</td><td>{$row[‘name’]}</td><td>{$row[‘surname’]}</td></tr>”;

}

?>

</table>

<?php

phpinfo();

?>

</body>

</html>

Напишем docker-compose.yml. Содержимое файла представлено в листинге 3.

Листинг 3 – docker-compose.yml

version: '3'

services:

  php:

    build:

      ./php

    ports:

      - 8080:80

    volumes:

      - ./www:/var/www/html

    depends\_on:

      - data

  data:

    image: mysql

    container\_name: DB

    volumes:

        - "./sql/init.sql:/docker-entrypoint-initdb.d/1.sql"

    environment:

        MYSQL\_ROOT\_PASSWORD: password

Переходим в рабочую директорию и с помощью команды docker-compose up собираем, создаем и запускаем контейнеры (рис.2).

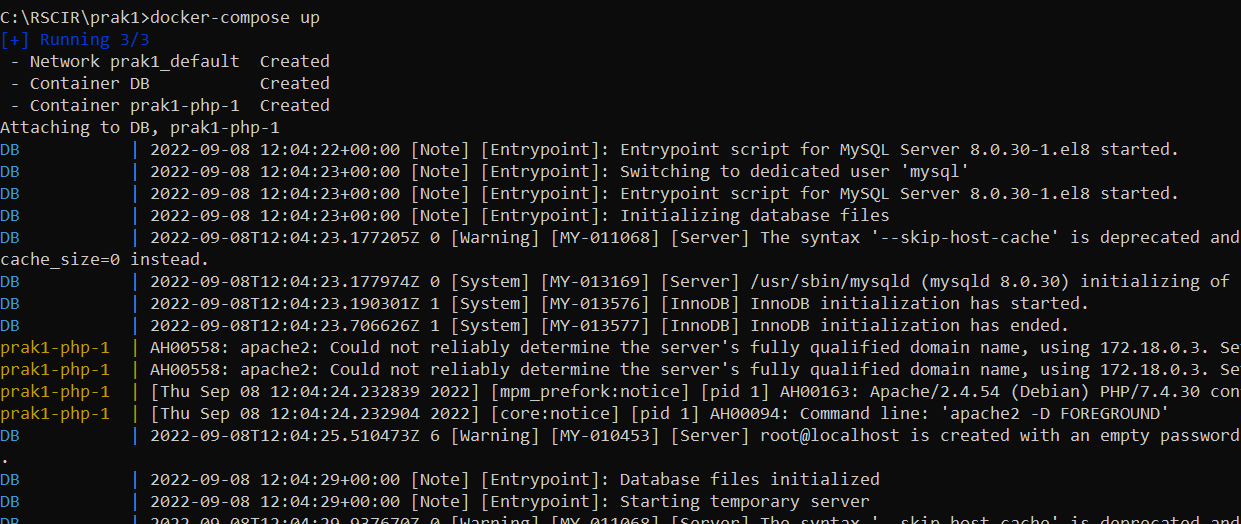


Рисунок 2 - Запуск контейнеров

Перейдем по адресу localhost:8080 и убедимся в работоспособности веб-сервера, отображающиего выборку из базы данных и информационное сообщение о версии языка программирования, его настройках и конфигурации (рис.3).

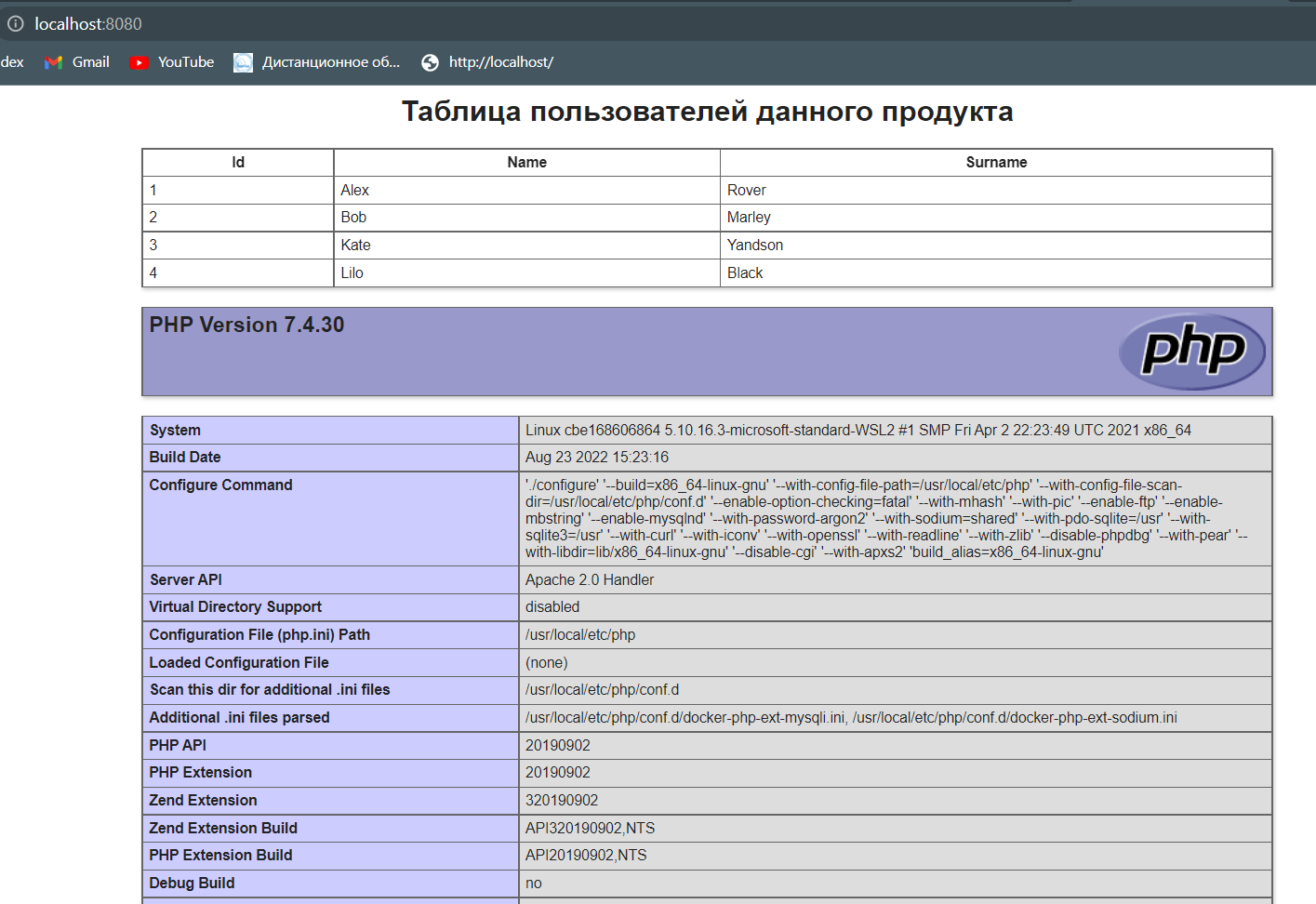


Рисунок 3 – Проверка конфигурации

Ответы на вопросы к практической:

**1. Сервер и клиент.**

Сервер (программное обеспечение) - программный компонент вычислительной системы, выполняющий сервисные (обслуживающие) функции по запросу клиента, предоставляя ему доступ к определённым ресурсам или услугам.

Сервер (аппаратное обеспечение) - выделенный или специализированный компьютер для выполнения сервисного программного обеспечения без непосредственного участия человека.

Клиент — это аппаратный или программный компонент вычислительной системы, посылающий запросы серверу.

**2. База данных.**

База данных — это информационная модель, позволяющая упорядоченно хранить данные об объекте или группе объектов, обладающих набором свойств, которые можно категорировать. Базы данных функционируют под управлением систем управления базами данных (сокращенно СУБД).

**3. API.**

API (Application Programming Interface - прикладной программный интерфейс) - набор функций и подпрограмм, обеспечивающий взаимодействие клиентов и серверов. API (в клиент-сервере) - описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

**4. Сервис, отличия от сервера.**

Сервис - легко заменяемый компонент сервисно-ориентированной архитектуры со стандартизированными интерфейсами.

Сервер находится уровнем выше, чем сервис. Любой полученный запрос прежде, чем попадёт в сервис, проходит через сервер. И уже сервер передаёт запрос сервису. И сервис отдаёт ответ именно серверу. А уже сервер отправляет ответ клиенту.

**5. Архитектура клиент-сервер.**

Данная модель — это идея разделения системы или приложения на отдельные задачи, размещаемые на различных платформах для большей эффективности. Уже применение данной идеи лежит в основе архитектуры клиент-сервер, распределенных вычислений, архитектуры приложений и т.д.

Клиент представляет собой программу представления данных, которая для их получения посылает запросы серверу, который в свою очередь может делать запрос к базе данных, обрабатывает данные и возвращает их к клиенту. Возможны случаи разделение обработки данных, когда часть работы сервера в виде обработки данных выполняет клиент. Но нужно понимать, что в этом случае очень важно разделение обязанностей и уровней доступа к данным на стороне клиента.

**6. Виды сервисов.**

Существует великое множество возможных сервисов, как самостоятельных, так и в составе приложений. Рассмотрим возможные виды сервисов:

* Серверы приложений;
* Веб-серверы;
* Серверы баз данных;
* Файл-серверы;
* Прокси-серверы;
* Файрволы (брандмауэры);
* Почтовые серверы.

**7. Масштабируемость.**

Масштабируемость - способность работать с увеличенной нагрузкой путем наращивания ресурсов без фундаментальной перестройки архитектуры и/или модели реализации при добавлении ресурсов.

Система называется масштабируемой, если она способна увеличивать производительность пропорционально дополнительным ресурсам. Основными являются горизонтальная и вертикальная масштабируемость.

**8. Протоколы передачи данных.**

Протокол передачи данных - набор определенных правил или соглашений интерфейса логического уровня, который определяет обмен данными между различными программами. Эти правила задают единообразный способ передачи сообщений и обработки ошибок.

**9. Тонкий и толстый клиенты.**

При классификации компонентов архитектуры клиент-сервер существует понятия “толстый” и “тонкий” клиент. При применении толстого клиента полная функциональность приложения обеспечивается вне зависимости от сервера. В данном случае сервер чаще всего выступает в роли хранилища информации, а вся логика приложения, как и механизм отображения данных располагаются и выполняются на клиенте. Тонким клиентом называют компьютеры и программы, функционирующие в терминальной или серверной сети. Множество задач по обработке данных осуществляются на главных компьютерах, к которым присоединено приложение и компьютер. Тонкий клиент же в отличие от толстого только отображает данные, принятые от сервера.

**10. Паттерн MVC: общие тезисы.**

Первая часть данного паттерна это модель (Model). Это представление содержания функциональной бизнес-логики приложения.

Представление (View) это есть отображение данных, получаемых от модели. Никакого влияния на модель представление оказать не может.

Третьим компонентом системы является контроллер. Данный компонент является неким буфером между моделью и представлением. Обобщенно он управляет представлением на основе изменения модели.

**11. Паттерн MVC: Model-View-Presenter.**

Особенностью паттерна Model-View-Presenter является то, что он позволяет создавать абстракцию представления. Для реализации данного метода выделяется интерфейс представления. А презентер получает ссылку на реализацию интерфейса, подписывается на события представления и по запросу меняет модель.

**12. Паттерн MVC: Model-View-View Model.**

Особенностью паттерна Model-View-View Model является связывание элементов представления со свойствами и событиями View-модели.

**13. Паттерн MVC: Model-View-Controller.**

Особенностью паттерна Model-View-Controller является то, что контроллер и представление зависят от модели, но при этом сама модель не зависит от двух других компонентов.

**14. Docker: общие тезисы и определения.**

Существует проблема разработки того или иного приложения и его развертывания на других машинах. Самыми частыми решениями данной проблемы является установочные скрипты, облачные сервисы и виртуальные машины. Описанные подходы не являются оптимальными что раздувает техническую поддержку до максимума, а также медленны и тяжеловесны. Одним из вариантов решения данной задачи является докер, который представляет технологию контейнеризации.

**15. Dockerfile.**

Часто возникает ситуация, когда конфигурации уже существующего не хватает. Чтобы создавать свои собственные образы нужен специальный скрипт. Образы наследуются и, обычно, для создания своего первого образа мы берём готовый образ и наследуемся от него. Чтобы запустить скрипт он должен иметь имя Dockerfile и не должен иметь типа.

Чтобы собрать новый образ, нужно, в папке, где находится Dockerfile, выполнить команду: “docker build -t my\_image .”. Тег -t задает название образа или тег, а точка обозначает работу в текущей директории.

**16. Docker Compose.**

Когда идет работа с несколькими контейнерами, то требуется механизм их объединения и оркестровки. Таким инструментом является Docker Compose. Это средство для решения задач развертывания проектов. Docker Compose используется для одновременного управления несколькими контейнерами, входящими в состав приложения. Этот инструмент предлагает те же возможности, что и Docker, но позволяет работать с более сложными приложениями.

**17. LAMP.**

Для полноценной работоспособности конфигурации нужны: операционная система, Веб-сервер, язык программирования и База данных. Из всего этого следует идея технологии LAMP — акроним, обозначающий набор (комплекс) серверного программного обеспечения, широко используемый в интернете. LAMP назван по первым буквам входящих в его состав компонентов: Linux, Apache, MySQL, PHP.

ВЫВОДЫ

В процессе выполнения практической работы мы создали свою конфигурацию серверного программного обеспечения, в которой присутствуют веб-сервер, операционная система, язык программирования и база данных. Для проверки работоспособности конфигурации мы сгенерировали тестовую страничку с необходимыми данными.