|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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»

**Студент группы** ИКБО-16-20 Косогоров Кирилл Станиславович

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

(подпись студента)

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

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

(подпись руководителя)

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

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

Москва 2022

Оглавление

[Цель работы: 2](#_Toc126012035)

[Постановка задачи: 3](#_Toc126012036)

[Ход работы: 3](#_Toc126012037)

[Вывод: 5](#_Toc126012038)

[Ответы на вопросы: 6](#_Toc126012039)

[Ссылка на репозиторий GitHub 11](#_Toc126012040)

[Список использованной литературы 11](#_Toc126012041)

# Цель работы:

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

# Постановка задачи:

Для проверки работоспособности 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 конфигурации требуется сгенерировать тестовую страничку, содержащую выборку из созданной таблицы и информационное сообщение о версии языка программирования, его настройках и конфигурации.

# Ход работы:

Мною было принято решение использовать технологии WAMP — Microsoft Windows, Apache, MySQL, PHP. Для этого необходимо было создать файл docker-compose.yml , в котором будет прописаны инструкции для запуска и настройки необходимых нам сервисов. Структура проекта будет выглядеть следующим образом: директория database, в котором будет хранится скрипт для создания и инициализирования базы данных, который был предоставлен в качестве проверки работоспособности конфигурации, директория server, в котором хранится index.php и style.css (Рисунок 1)

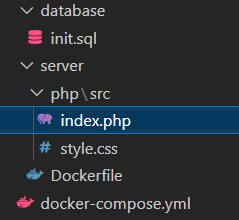


Рисунок 1 – Структура проекта.

Также в директории server присутствует Dockerfile, необходимый для закачки образа php apache и установки нужных расширений (Рисунок 2).

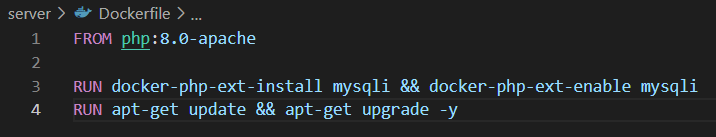


Рисунок 2 – Содержимое Dockerfile.

В файле docker-compose.yml прописаны инструкции для запуска необходимых нам технологий. На рисунке 3 прописаны инструкции для MYSQL базы данных и для php-apache.

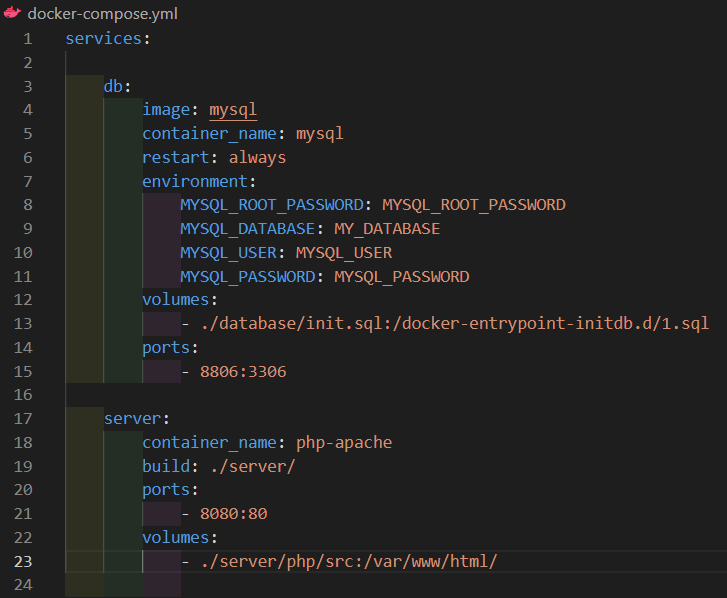


Рисунок 3 – Инструкции для базы данных.

Через консоль осуществим запуск с помощью команды docker-compose up (Рисунок 4).

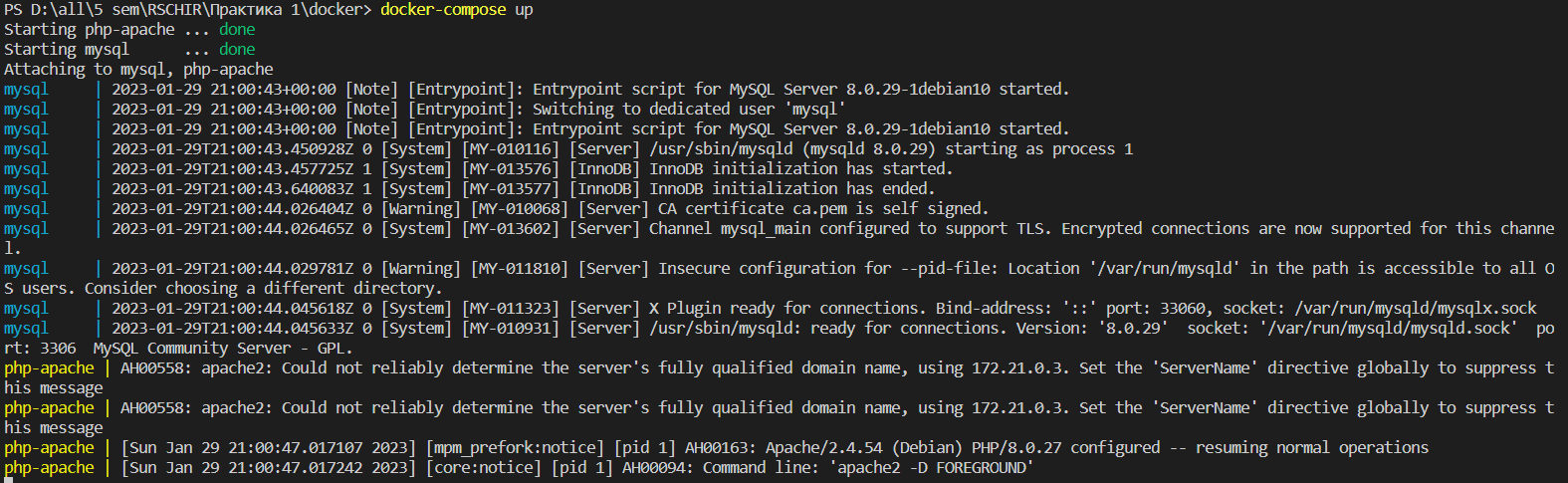


Рисунок 4 – Запуск docker-compose.

Переходим по localhost:8080 и видим наш готовый результат (Рисунок 5).

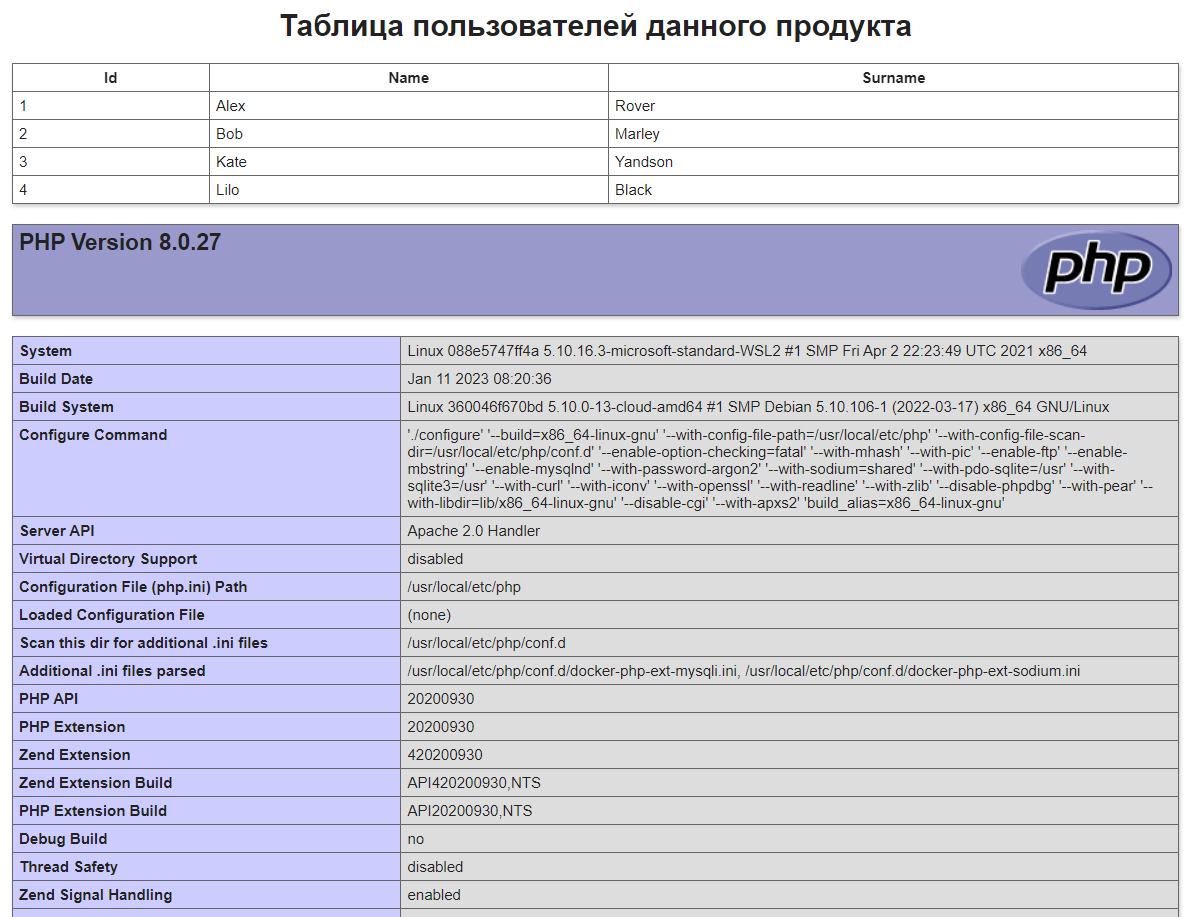


Рисунок 5 – Результат.

# Вывод:

В ходе выполнения практической работы была успешна установлена конфигурация WAMP и выполнены поставленные задачи.

# Ответы на вопросы:

1. Сервер и клиент.

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.

1. 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).

1. 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.

1. Сервис, отличия от сервера.

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

Сервис (микросервис) - это программа, размещаемая на сервере, которая предоставляет определенную функцию, находясь по определенному адресу и используя четко определенные интерфейсы и логику. Сегодня это чаще всего интерфейсы API.

Сервер может хостить в себе тысячи сервисов. Впрочем, и один сервис может быть размещен на нескольких серверах. В более широком плане - сервер - это виртуальная машина, которая хостит в себе Apache, IIS и много-много web-сервисов. Еще в более широком плане - это железяка, которая хостит в себе множество виртуалок.

1. Архитектура клиент-сервер.

Архитектура «Клиент-Сервер» предусматривает разделение процессов предоставление услуг и отправки запросов на них на разных компьютерах в сети, каждый из которых выполняют свои задачи независимо от других.

В архитектуре «Клиент-Сервер» несколько компьютеров-клиентов (удалённые системы) посылают запросы и получают услуги от централизованной служебной машины – [сервера](https://itelon.ru/catalog/oborudovanie/servers/), которая также может называться хост-системой.

1. Виды сервисов.

* Серверы приложений
* Веб-серверы
* Серверы баз данных
* Файл-серверы
* Прокси-сервер
  + Веб-прокси
  + Обратный прокси
* Файрволы
* Почтовые серверы

7. Масштабируемость.

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

8. Протоколы передачи данных.

* TCP/IP – Transmission Control Protocol/Internet Protocol
* UDP – User Datagram Protocol
* IP – Internet Protocol
* FTP – File Transfer Protocol
* DNS – Domain Name System
* HTTP – HyperText Transfer Protocol
* NTP – Network Time Protocol
* SSH – Secure SHell

9. Тонкий и толстый клиенты.

Толстый клиент — клиент, выполняющий запрашиваемые со стороны пользователя манипуляции независимо от ведущего сервера. Основной сервер в такой вариации системной архитектуры может применяться как особое хранилище информации, обработка и конечное предоставление которых просто переносится на локальную машину пользователя.

Тонкий клиент — вид клиента, который может переносить выполнение задач по обработке информации на сервер, не применяя свои мощности по вычислению для их внедрения. Все вычислительные ресурсы подобного клиента максимально ограничены, важно, чтобы их хватало для старта нужного сетевого ПО, применяя, к примеру, веб-интерфейс.

10. Паттерн MVC: общие тезисы.

Model, View, Controller (MVC) — шаблон (паттерн) программирования, разделяющий архитектуру приложения на три модуля: модель (Model), представление (View), контроллер (Controller). Он позволяет изменять каждый компонент независимо друг от друга для простой разработки и поддержки веб-приложений.

* **Модель (Model).** Это основная логика приложения. Отвечает за данные, методы работы с ними и структуру программы. Модель реагирует на команды из контроллера и выдает информацию и/или изменяет свое состояние. Она передает данные в представление.
* **Представление (View).** Задача компонента — визуализация информации, которую он получает от модели. View отображает данные на уровне пользовательского интерфейса. Например, в виде таблицы или списка. Представление определяет внешний вид приложения и способы взаимодействия с ним.
* **Контроллер (Controller).** Он обеспечивает взаимодействие с системой: обрабатывает действия пользователя, проверяет полученную информацию и передает ее модели. Контроллер определяет, как приложение будет реагировать на действия пользователя. Также контроллер может отвечать за фильтрацию данных и авторизацию.

11. Паттерн MVC: Model-View-Presenter.

MVP (Model-View-Presenter) — паттерн разработки пользовательского интерфейса. Шаблон MVP является производным от MVC, но при этом имеет несколько иной подход. Основное отличие — представление (presenter) не так сильно связано моделью (model).

Функциональные части MVP:

* Model  
  Модель представляет данные для отображения пользователю.
* View  
  Вид реализует отображение данных, представленных моделью, а также взаимодействует с представлением для обновлений.
* Presenter  
  Представление содержит основную бизнес-логику, так как взаимодействует со всеми частями модели.

**Основное отличие MVP и MVC** в том, что в MVC обновлённая модель сама говорит виду, что нужно показать другие данные. Если же этого не происходит и приложению нужен посредник в виде представителя, то паттерн стоит называть MVP.



12. **MVVM (Model-View-ViewModel)** — способ организации кода. Он также помогает отделить пользовательский интерфейс от логики. Данный паттерн во многом схож с MVP

* **View** содержит только код пользовательского интерфейса. Он отображает экран (текстовые поля, ярлыки, кнопки) и пользовательский ввод.
* **Model** — это классы. Это «вещи» в вашей программе. Эти классы будут содержать логику, необходимую для выполнения функций.

**• ViewModel,** как и Presenter в MVP, является посредником между View и моделью. Но ViewModel не может напрямую воздействовать на View, а лишь является источником данных и имеет возможность через функции вызова передавать актуальные данные. То, что отображать, определяется на уровне View.

14. Docker — программное обеспечение для автоматизации развёртывания и управления приложениями, контейнеризатор приложений.

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 медленны и тяжеловесны. Одним из вариантов решения данной задачи является докер, который представляет технологию контейнеризации. Подобно виртуальной машине докер запускает свои процессы в собственной, заранее настроенной операционной системе. Но при этом все процессы докера работают на физическом host-сервере, деля все процессоры и всю доступную память со всеми другими процессами, запущенными в hostсистеме. Подход, используемый Docker, находится посередине между запуском всего на физическом сервере и полной виртуализацией, предлагаемой виртуальными машинами. Этот подход называется контейнеризацией.

Основными компонентами докера является:

● docker daemon — сердце докера. Это демон, работающий на хост-машине, и умеющий сохранять с удалённого репозитория и загружать на него образы, запускать из них контейнеры, следить за запущенными контейнерами, настраивать сеть между контейнерами (а с версии 0.8 и между машинами). А еще именно демон создает образы контейнеров.

● docker — это консольная утилита для управления docker-демоном по HTTP. Она устроена очень просто и работает предельно быстро. Управлять демоном докера можно откуда угодно, а не только с той же машины.

● docker Hub централизованно хранит образы контейнеров. Когда вы пишете “docker run ruby”, docker скачивает самый свежий образ с ruby именно из публичного репозитория.

● Docker-образ — шаблон для создания Docker-контейнеров. Представляет собой исполняемый пакет, содержащий все необходимое для запуска приложения: код, среду выполнения, библиотеки, переменные окружения и файлы конфигурации.

● Контейнер — это экземпляр образа во время выполнения. Он состоит из образа, среды выполнения и стандартного набора инструкций.

15. Dockerfile — это инструкция для сборки образа. Это простой текстовый файл, содержащий по одной команде в каждой строке. В нем указываются все программы, зависимости и образы, которые нужны для разворачивания образа.

Для примера рассмотрим Dockerfile:

FROM python:3

COPY main.py /

CMD ["python", "./main.py" ]

Первая строчка означает, что за основу мы берем образ с названием python версии 3. Docker найдет его в docker registry, скачает и будет использовать за основу. Вторая строчка означает, что нужно скопировать файл *main.py* в корень файловой системы контейнера. Третья строчка означает, что нужно запустить python и передать ему в качестве параметра название файла *main.py*.

16. Когда идет работа с несколькими контейнерами, то требуется механизм их объединения. Таким инструментом является Docker Compose. Это средство для решения задач развертывания проектов. Docker-compose позволяет развернуть и настроить все приложения одной командой, а без него пришлось бы разворачивать и настраивать каждый контейнер отдельно.

17. 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 — операционная система Linux;

● Apache — веб-сервер;

● MariaDB / MySQL — СУБД;

● PHP — язык программирования, используемый для создания вебприложений (помимо PHP могут подразумеваться другие языки, такие как Perl и Python).

Содержание данного набора может варьироваться в зависимости от задач и технического задания

# Ссылка на репозиторий GitHub

<https://github.com/pe4enka2515/DCPIR/tree/main/%D0%9F%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%BA%D0%B0%201/docker>

# Список использованной литературы

1. Конспект лекции №1 (с сайта ЦДО) [Электронный ресурс] – URL: <https://online-edu.mirea.ru/pluginfile.php?file=%2F955443%2Fmod_resource%2Fcontent%2F1%2FКонспект%20лекции%20№1.pdf> (Дата обращения 08.09.2022)