|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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-13-20 Костерев Е.Д.

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

(подпись студента)

**Руководитель практической работы** ассистент Мельников Д.А.

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

(подпись руководителя)

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

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

Москва 2022

СОДЕРЖАНИЕ

[Цель работы: 4](#_Toc113911143)

[Ход работы: 4](#_Toc113911144)

[ВЫВОД 8](#_Toc113911145)

[Ответы на вопросы к практической работе 9](#_Toc113911146)

[1. Сервер и клиент 9](#_Toc113911147)

[2. База данных 9](#_Toc113911148)

[3. API 9](#_Toc113911149)

[4. Сервис, отличие от сервера 9](#_Toc113911150)

[5. Архитектура клиент-сервер 10](#_Toc113911151)

[6. Виды сервисов 10](#_Toc113911152)

[7. Масштабируемость 12](#_Toc113911153)

[8. Протоколы передачи данных 12](#_Toc113911154)

[9. Тонкий и толстый клиенты 12](#_Toc113911155)

[10. Паттерн MVC: общие тезисы 13](#_Toc113911156)

[11. Паттерн MVC: Model-View-Presenter 13](#_Toc113911157)

[12. Паттерн MVC: Model-View-View Model 14](#_Toc113911158)

[13. Паттерн MVC: Model-View-Controller 14](#_Toc113911159)

[14. Docker: общие тезисы и определения 15](#_Toc113911160)

[15. Dockerfile 16](#_Toc113911161)

[16. Docker Compose 16](#_Toc113911162)

[17. LAMP 16](#_Toc113911163)

[Ссылка на удаленный репозиторий проекта 17](#_Toc113911164)

[Список использованной литературы 18](#_Toc113911165)

**Цель работы:**

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

## Ход работы:

1. Для проверки работоспособности конфигурации, были скачены скрипты SQL и PHP (Рисунок 1, 2).

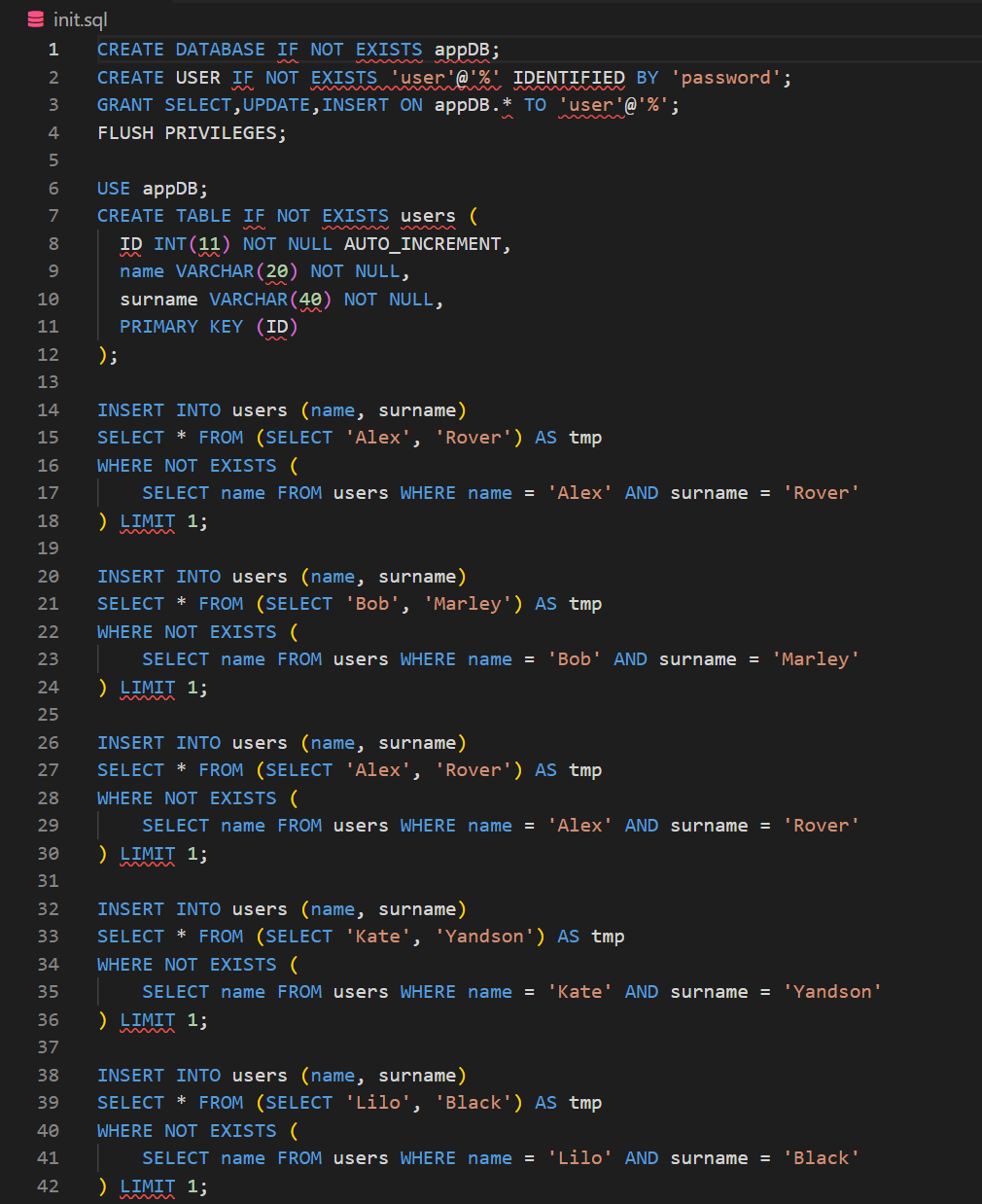


Рисунок 1 — Скрипт на SQL



Рисунок 2 — Скрипт на SQL

1. Также был написан Dockerfile и файл docker-compose (Рисунок 3, 4).

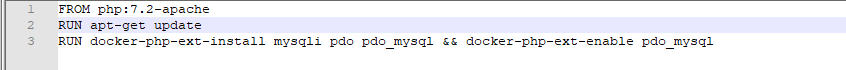


Рисунок 3 — Dockerfile

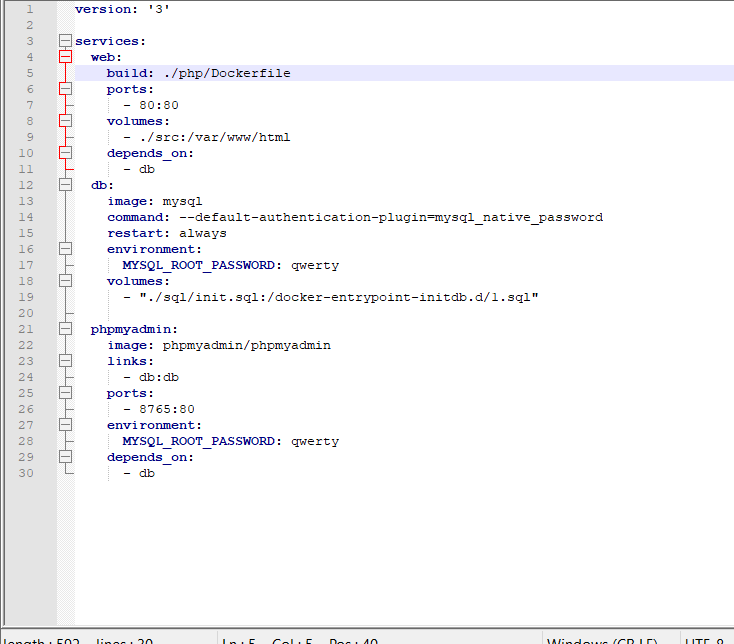


Рисунок 4 — Docker-compose

1. Дальше нужно собрать и запустить docker-compose, при помощи которого запускается несколько контейнеров одновременно (Рисунок 5).

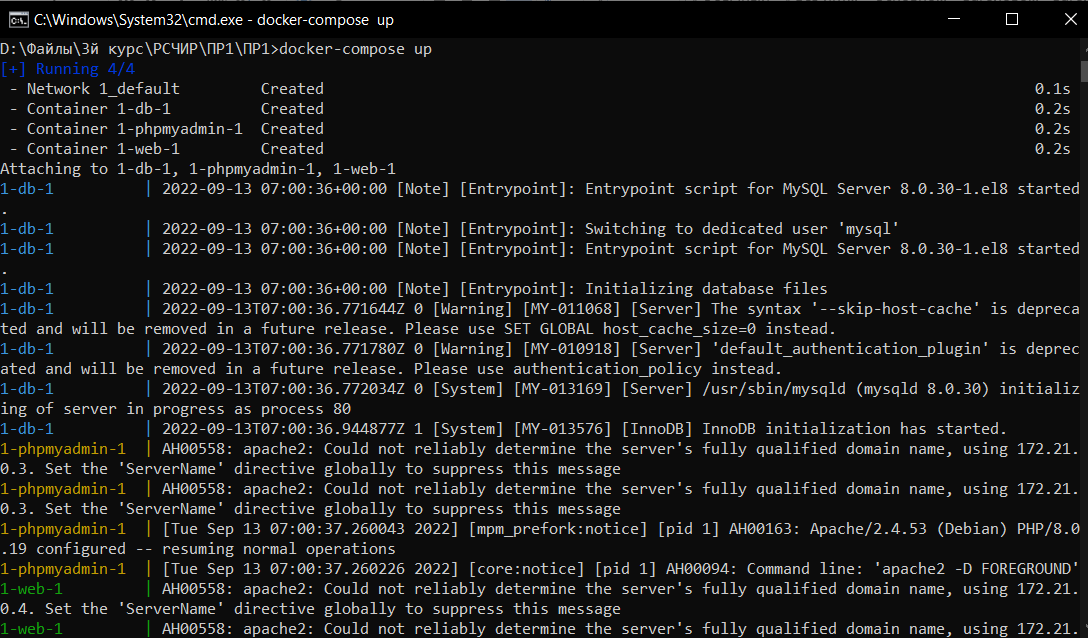


Рисунок 5 — Сборка проекта

1. Подключаемся к порту (Рисунок 6).

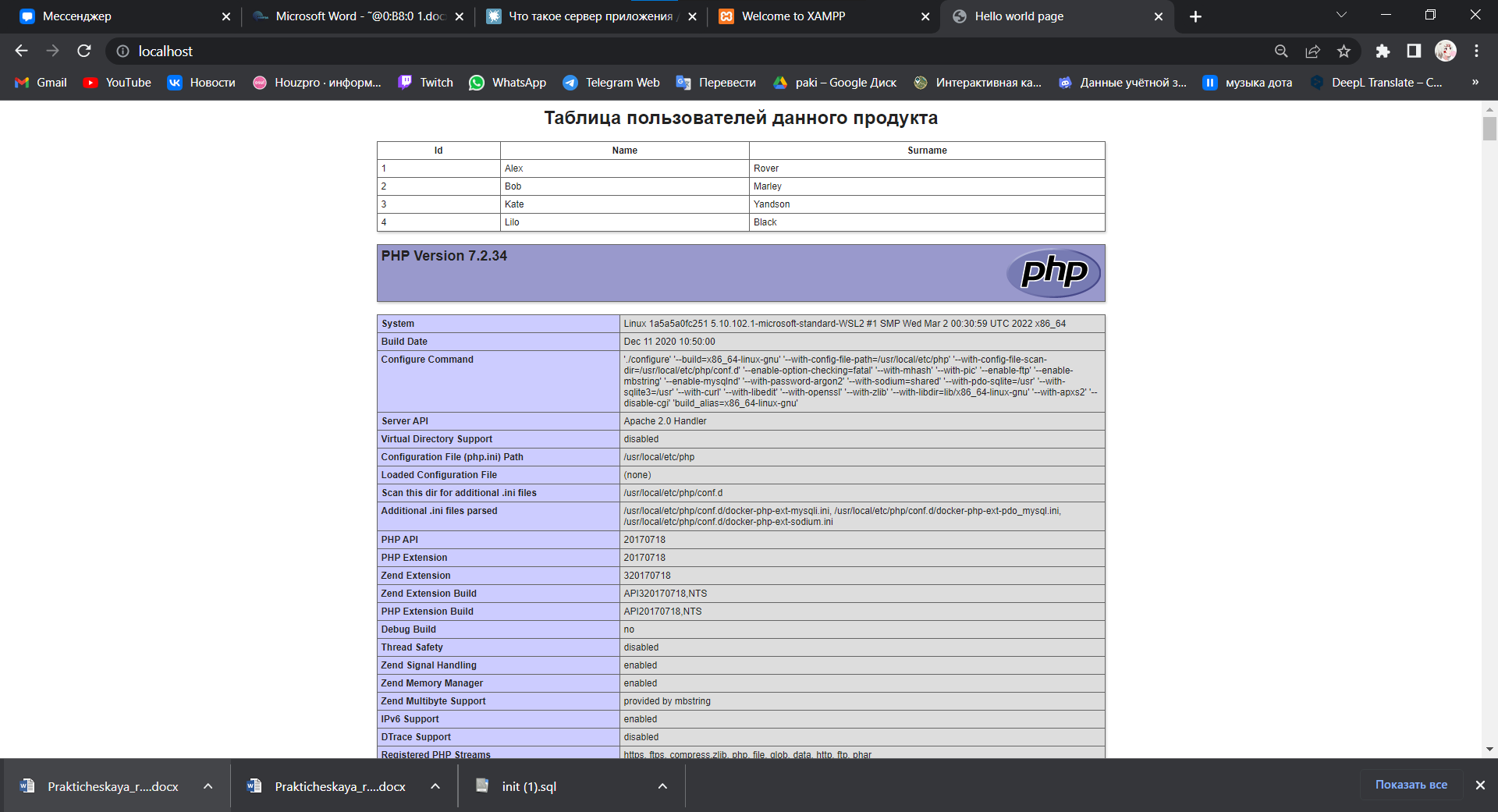


Рисунок 6 — Полученная страничка

# ВЫВОД

В ходе практической работы была создана конфигурация серверного программного обеспечения, в которой должны присутствовать веб-сервер, операционная система, язык программирования и база данных.

# Ответы на вопросы к практической работе

## Сервер и клиент

Сервер и клиент (рабочая станция) могут иметь одинаковую аппаратную конфигурацию, так как различаются лишь по участию в своей работе человека.

## 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)

## 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.

## Сервис, отличие от сервера

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.

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

## Архитектура клиент-сервер

Архитектура клиент-сервер — это сетевое окружение, в котором управление данными осуществляется на серверном узле, а другим узлам (клиентам) предоставляется доступ к данным. К достоинствам клиент-серверной архитектуры можно отнести: ● основная нагрузка ложится на сервер(а), в системе имеется одна (или несколько) мощная(ых) серверная(ых) конфигурация часто сложной конструкции и множество “слабых” машин клиентов, что снижает общую стоимость всей системы; ● данные находятся в безопасности, так как сервер дает клиенту только требуемую выборку данных для клиента, основываясь на его уровне доступа; ● разграничение полномочий между клиентом и сервером; ● кроссплатформенность - реализаций клиента может быть сколько угодно: от веб-браузера до приложений мобильных платформ; ● нет дублирования сервисов на каждом клиенте, основная обработка данных лежит на сервере.

## Виды сервисов

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

● Серверы приложений.

Сервер приложений (англ. application server) — это программная платформа (фреймворк), предназначенная для эффективного исполнения процедур (программ, скриптов), на которых построены приложения.

● Веб-серверы.

Являются подвидом серверов приложений. Изначально предоставляли доступ к гипертекстовым документам по протоколу HTTP. Сейчас поддерживают расширенные возможности, в частности, передачу произвольных данных.

● Серверы баз данных.

Серверы баз данных используются для обработки запросов. На сервере находится СУБД для управления БД и ответов на запросы.

● Файл-серверы.

Файл-сервер хранит информацию в виде файлов и предоставляет пользователям доступ к ней. Как правило, файл-сервер обеспечивает и определенный уровень защиты от несанкционированного доступа.

● Прокси-сервер.

Прокси-сервер (от англ. proxy - представитель, уполномоченный; часто просто прокси, сервер-посредник) - промежуточный сервер (комплекс программ) в компьютерных сетях, выполняющий роль посредника. Существует несколько видов прокси-серверов:

● Веб-прокси — широкий класс прокси-серверов, служащий для кэширования данных.

● Обратный прокси — прокси-сервер, который, в отличие от вебпрокси, ретранслирует запросы клиентов из внешней сети на один или несколько серверов, логически расположенных во внутренней сети. Часто используется для балансировки сетевой нагрузки между несколькими веб-серверами и повышения их безопасности, играя при этом роль межсетевого экрана на прикладном уровне.

● Файрволы (брандмауэры).

Межсетевые экраны, анализирующие и фильтрующие проходящий сетевой трафик, с целью обеспечения безопасности сети.

● Почтовые серверы.

Предоставляют услуги по отправке и получению электронных почтовых сообщений.

## Масштабируемость

Вертикальная масштабируемость(англ. scaling up) представляет собой увеличение производительности компонентов серверной системы в интересах повышения производительности всей системы. Следует понимать, что система ограничена объемом памяти, скоростными характеристиками и т.д. Данный метод не снимает нагрузку на всю систему, но является самым простым. Ярким примером является увеличение оперативной памяти, установка более мощного процессора.

Горизонтальная масштабируемость(англ. scaling out) представляет собой как разбиение системы на более мелкие структурные компоненты и разнесение их, так и увеличение количества компонентов, параллельно выполняющих одну и ту же функцию. Данный метод часто требует изменение программного обеспечения для налаживания взаимодействия между новыми компонентами. Частым примером является добавление еще одного сервера тех же характеристик к существующему.

## 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.

## 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. Даже при отсутствии соединения с сервером работа ведется с локальными копиями данных, а при возобновлении соединения происходит синхронизация данных.

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. Вся логика приложения выполняется на более производительном сервере, что не требует клиентских мощностей, кроме хорошего и стабильного канала связи. К сожалению, любой сбой на сервере и в канале связи влечет “падение” всего приложения.

## Паттерн MVC: общие тезисы

Существуют разные виды MVC-паттерна, различающихся 3 компонентом системы. Наиболее распространенными видами являются:

● Model-View-Presenter;

● Model-View-View Model;

● Model-View-Controller.

## 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.

Признаки подхода с использованием презентера:

● двусторонняя коммуникация с представлением;

● представление взаимодействует напрямую с презентером, путем вызова соответствующих функций или событий экземпляра презентера;

● презентер взаимодействует с View путем использования специального интерфейса, реализованного представлением;

● одному презентеру соотвествует одно отображение.

## Паттерн MVC: Model-View-View Model

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

Признаками данного подхода являются:

● Двусторонняя коммуникация с представлением.

● View-модель — это абстракция представления. Означает, что свойства представления совпадают со свойствами View-модели / модели.

● View-модель не имеет ссылки на интерфейс представления (IView). Изменение состояния View-модели автоматически изменяет представление и наоборот, поскольку используется механизм связывания данных (Bindings).

● Одному экземпляру View-модели соответствует одно отображение.

## 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.

Признаками данного подхода являются:

● Контроллер определяет, какое представление должно быть отображено в требуемый момент. Если рассматривать применение для разработки вебприложений, то контроллер управляет запросами пользователя. Его основная функция — вызывать и координировать действие необходимых ресурсов и объектов, нужных для выполнения действий, задаваемых пользователем. Обычно контроллер вызывает соответствующую модель для задачи и выбирает подходящий вид.

● События представления могут повлиять только на контроллер. Контроллер может повлиять на модель и определить другое представление.

● Возможно несколько представлений только для одного контроллера. Данный вариант чаще всего используется при разработке веб-приложений.

## Docker: общие тезисы и определения

Docker — это платформа, которая позволяет упаковать в контейнер приложение со всем окружением и зависимостями, а затем доставить и запустить его в целевой системе.

Приложение, упакованное в контейнер, изолируется от операционной системы и других приложений. Поэтому разработчики могут не задумываться, в каком окружении будет работать их приложение, а инженеры по эксплуатации — единообразно запускать приложения и меньше заботиться о системных зависимостях.

В поставку Docker входят следующие компоненты:

Docker host — это операционная система, на которую устанавливают Docker и на которой он работает.

Docker daemon — служба, которая управляет Docker-объектами: сетями, хранилищами, образами и контейнерами.

Docker client — консольный клиент, при помощи которого пользователи взаимодействуют с Docker daemon и отправляют ему команды, создают контейнеры и управляют ими.

Docker image — это неизменяемый образ, из которого разворачивается контейнер.

Docker container — развёрнутое и запущенное приложение.

Docker Registry — репозиторий, в котором хранятся образы.

Dockerfile — файл-инструкция для сборки образа.

Docker Compose — инструмент для управления несколькими контейнерами. Он позволяет создавать контейнеры и задавать их конфигурацию.

Docker Desktop — GUI-клиент, который распространяется по [GPL](https://skillbox.ru/media/code/gayd-po-svobodnym-litsenziyam-ot-gnu-chto-eto-takoe-i-kakimi-oni-byvayut/?ysclid=l2hyfvpod2). Бесплатная версия работает на Windows, macOS, а с недавних пор и на Linux. Это очень удобный клиент, который отображает все сущности Docker и позволяет запустить однонодовый Kubernetes для компьютера.

## Dockerfile

Dockerfile — это текстовый файл с инструкциями, необходимыми для создания образа контейнера. Эти инструкции включают идентификацию существующего образа, используемого в качестве основы, команды, выполняемые в процессе создания образа, и команду, которая будет выполняться при развертывании новых экземпляров этого образа контейнера.

## Docker Compose

Docker-compose — это надстройка над докером, приложение написанное на Python, которое позволяет запускать множество контейнеров одновременно и маршрутизировать потоки данных между ними.

## LAMP

Docker-LAMP - это набор образов docker, которые включают базовое изображение phusion (варианты 14.04, 16.04 и 18.04), а также стек LAMP (Apache, MySQL и PHP) в одном удобном пакете.

# Ссылка на удаленный репозиторий проекта

<https://github.com/houzpro/RSCHIR>

# Список использованной литературы

1. «Лекция 1. Архитектура серверных частей интернет-ресурсов.», 2021 – 12 стр.
2. Docker для начинающих [Электронный ресурс]. Режим доступа: https://www.youtube.com/watch?v=xuO8zv62HXM, свободный.
3. dockerhub [Электронный ресурс]. <https://hub.docker.com/r/fauria/lamp> свободный.