**Вопросы к практической работе.**

1. Сервер и клиент.

2. База данных.

3. API.

4. Сервис, отличия от сервера.

5. Архитектура клиент-сервер.

6. Виды сервисов.

7. Масштабируемость.

8. Протоколы передачи данных.

9. Тонкий и толстый клиенты.

10. Паттерн MVC: общие тезисы.

11. Паттерн MVC: Model-View-Presenter.

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

13. Паттерн MVC: Model-View-Controller.

14. Docker: общие тезисы и определения.

15. Dockerfile

. 16. Docker Compose

. 17. LAMP

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.

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

# 2. **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** (Application Programming Interface - прикладной программный интерфейс) - набор функций и подпрограмм, обеспечивающий взаимодействие клиентов и серверов.

# 3.1 **API (в клиент-сервере)** - описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

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

# 4.1 **Веб-сервис(веб-служба**) - идентифицируемая уникальным веб-адресом (URL-адресом) программная система со стандартизированными интерфейсами.

# 5. 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**

* **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.

# 7 **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.

* **Вертикальная масштабируемость**(англ. scaling up) представляет собой увеличение производительности компонентов серверной системы в интересах повышения производительности всей системы. 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. Частым примером является добавление еще одного сервера тех же характеристик к существующему.

# 8 **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.

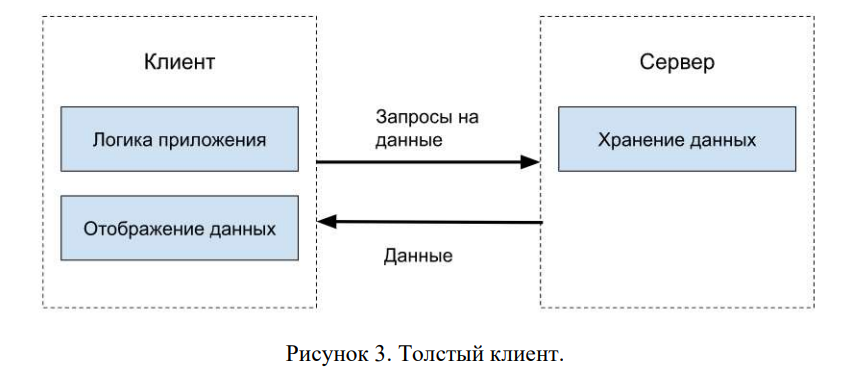
Наиболее известные прикладные протоколы, используемые в сети Интернет:

* Протокол RTP (Real-time Transport Protocol), протокол работает на прикладном уровне (OSI - 7) и используется при передаче трафика реального времени.
* HTTP (Hyper Text Transfer Protocol) — это протокол передачи гипертекста.
* FTP (File Transfer Protocol) — это протокол передачи файлов со специального файлового сервера на компьютер пользователя.
* POP3 (Post Office Protocol) — это стандартный протокол почтового соединения.
* SMTP (Simple Mail Transfer Protocol) — протокол, который задает набор правил для передачи почты.
* TELNET — это протокол удаленного доступа

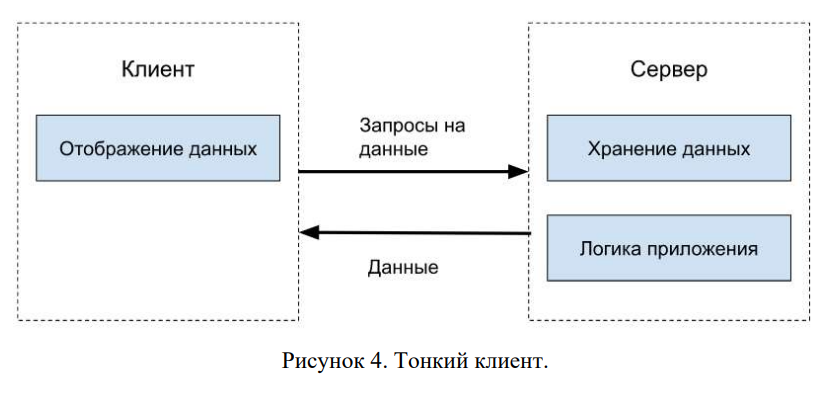
# 9 **Тонкий и толстый клиенты**

**Толстый клиент**

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.



**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



# 10 **MVC: общие тезисы**.

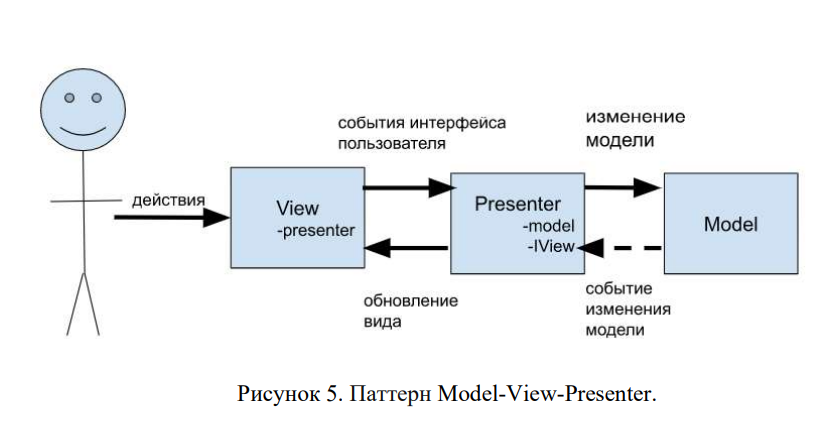
Название паттерну дают первые буквы его основных компонентов: Model View Controller.

**Первая часть данного паттерна это модель (Model**). Это представление содержания функциональной бизнес-логики приложения. Модель, как и любой компонент архитектуры под управлением данного паттерна не зависит от остальных частей продукта. То есть слой, содержащий модель ничего не знает о элементах дизайна и любом другом отображении пользовательского интерфейса

**Представление (View)** это есть отображение данных, получаемых от модели. Никакого влияния на модель представление оказать не может. Данное разграничение является разделением компетенций компонентов приложения и влияет на безопасность данных. Если рассматривать интернет-ресурсы представлением является html-страница.

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

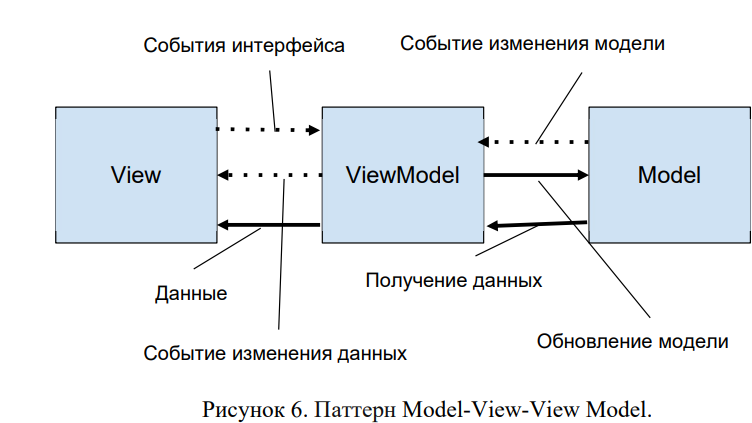
# 11 **Model-View-Presenter**

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.

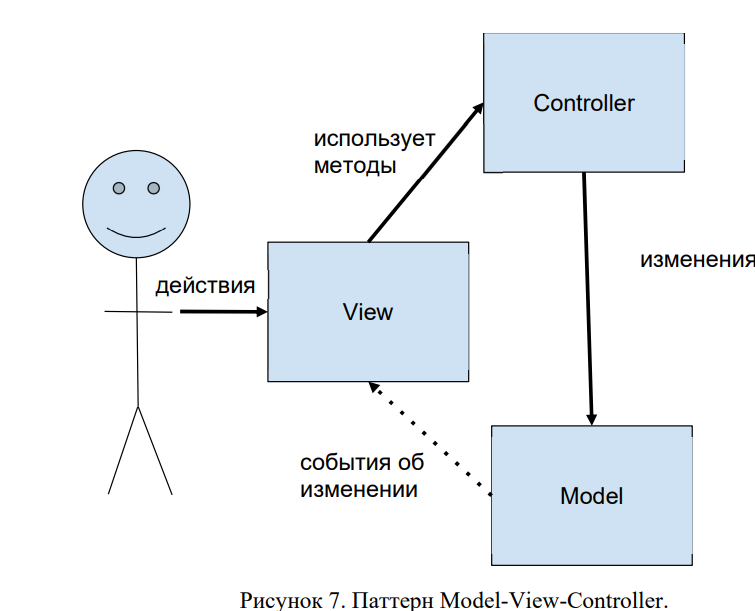
# 12 **Model-View-View Model**

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.

# 13 **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

# **14 Docer общие тезисы и определения**

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, собирать логи и настраивать сеть между контейнерами (а с версии 0.8 и между машинами). А еще именно демон создает образы контейнеров, хоть и может показаться, что это делает docker-client.
* docker — это консольная утилита для управления docker-демоном по HTTP. Она устроена очень просто и работает предельно быстро. Вопреки заблуждению, управлять демоном докера можно откуда угодно, а не только с той же машины. В сборке нового образа консольная утилита docker принимает пассивное участие: архивирует локальную папку в tar.gz и передает по сети docker-daemon, который и делает всю работу. Именно из-за передачи контекста демону по сети, лучше собирать тяжелые образы локально.
* docker Hub централизованно хранит образы контейнеров. Когда вы пишете “docker run ruby”, docker скачивает самый свежий образ с ruby именно из публичного репозитория. Изначально хаба не было, его добавили уже после очевидного успеха первых двух частей.
* Docker registry предназначен для хранения и дистрибуции готовых образов.

# 15. **Dockerfile**.

В файлах Dockerfile содержатся инструкции по созданию образа. С них, набранных заглавными буквами, начинаются строки этого файла. После инструкций идут их аргументы. Инструкции, при сборке образа, обрабатываются сверху вниз.

# 16 **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**

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).