**1. Система контроля версий. Виды систем контроля версий. Примеры систем контроля версий**

Система контроля версий — это программный инструмент, предназначенный для отслеживания изменений в файлах, координации совместной работы над проектами и обеспечения возможности отката к предыдущим версиям.

**Назначение:**

* Хранение всей истории изменений;
* Возможность восстановить предыдущую версию;
* Совместная работа над проектом с отслеживанием вклада каждого участника;
* Управление конфликтами при параллельном редактировании.

**Виды систем контроля версий:**

* **Локальные** Вся история изменений хранится на одном компьютере. Работа возможна только на локальной машине. Пример: RCS (Revision Control System). Недостаток — нет поддержки командной работы.
* **Централизованные (CVCS)** Существует единый центральный сервер, на котором хранится основная копия репозитория. Пользователи получают только рабочую копию. Пример: Subversion (SVN), Perforce. Плюсы — централизованное управление; минусы — зависимость от сервера, слабая автономность.
* **Распределённые (DVCS)** У каждого участника есть полная копия всего репозитория (включая историю). Работа возможна в оффлайне, а синхронизация производится позже. Примеры: Git, Mercurial. Преимущества — высокая надёжность, гибкость, возможность параллельной разработки.

**Наиболее популярная система сегодня — Git**, разработанная для управления исходным кодом в больших распределённых проектах (например, Linux).

**2. HTTP- и HTTPS-протоколы. Методы HTTP-запроса. HTTP заголовки. Группы кодов состояния при выполнении запросов**

**HTTP (HyperText Transfer Protocol)** — это протокол передачи данных в интернете, используемый для взаимодействия между клиентом (обычно браузером) и сервером. Он работает на уровне приложений и не обеспечивает шифрования данных.

**HTTPS (HTTP Secure)** — это расширение HTTP, в котором используется **шифрование по протоколам SSL/TLS** для защиты передаваемых данных. Обеспечивает:

* конфиденциальность (данные недоступны третьим лицам),
* целостность (невозможно незаметно изменить),
* аутентификацию сервера (через сертификаты).

**Методы HTTP-запросов:**

* **GET** — получение данных;
* **POST** — отправка данных на сервер (например, формы);
* **PUT** — обновление существующего ресурса;
* **DELETE** — удаление ресурса;
* **PATCH** — частичное обновление;
* **HEAD** — только заголовки, без тела ответа (для проверки существования ресурса).

**HTTP-заголовки:** Используются для передачи служебной информации. Примеры:

* Content-Type — тип содержимого (application/json, text/html);
* Authorization — данные авторизации (например, токен);
* Cache-Control — политика кэширования;
* User-Agent — идентификатор клиента;
* Accept — список допустимых форматов ответа.

**Коды состояния HTTP:**

* **1xx** — информационные (например, 100 Continue);
* **2xx** — успешные ответы (200 OK, 201 Created);
* **3xx** — перенаправления (301 Moved Permanently, 302 Found);
* **4xx** — ошибки клиента (403 Forbidden, 404 Not Found);
* **5xx** — ошибки сервера (500 Internal Server Error, 502 Bad Gateway).

**3. Клиент-серверная архитектура: назначение блоков, описание технических устройств клиентской и серверной части, описание связи базы данных с интерфейсом**

Клиент-сервер — это архитектурная модель, в которой взаимодействие происходит между двумя сторонами:

* **Клиент** инициирует запрос;
* **Сервер** обрабатывает его и отправляет ответ.

**Назначение:**

* Централизованное хранение и обработка данных;
* Упрощение масштабирования и обновления логики;
* Разделение ролей: интерфейс (клиент) и логика + хранение (сервер).

**Клиентская часть (frontend):**

* Это пользовательское устройство (ПК, смартфон) и программное обеспечение (браузер, приложение), через которое осуществляется взаимодействие;
* Отвечает за интерфейс, отправку запросов и отображение данных;
* Примеры устройств: ноутбуки, планшеты, терминалы, веб-браузеры.

**Серверная часть (backend):**

* Это вычислительный узел (физический сервер или облако), обрабатывающий запросы от клиентов;
* Реализует бизнес-логику, обеспечивает безопасность, доступ к базе данных и хранение информации;
* Сервер может быть специализированным (например, API-сервер, файловый сервер, сервер баз данных).

**Связь с базой данных:**

* Серверная часть взаимодействует с базой данных напрямую через SQL или другие запросы;
* Данные извлекаются, обрабатываются, и затем отправляются клиенту;
* Клиент никогда не взаимодействует с базой данных напрямую — только через сервер.

Таким образом, клиент отображает и запрашивает информацию, сервер принимает запрос, обрабатывает его с учётом бизнес-логики, обращается к базе данных, и возвращает результат.

**4. Модель TCP/IP: назначение уровней, протоколы. Маршрутизация: назначение, классификация, функции**

**Модель TCP/IP** — это концептуальная модель, описывающая, как данные передаются по сети. Она состоит из четырёх уровней:

* **Прикладной уровень** Отвечает за взаимодействие пользователя с сетью. Протоколы этого уровня работают напрямую с прикладными данными. Примеры:
  + **HTTP** — для веб-страниц
  + **SMTP** — для отправки электронной почты
  + **FTP** — для передачи файлов Здесь формируются запросы и ответы, видимые пользователю.
* **Транспортный уровень** Обеспечивает доставку данных от приложения к приложению на конечных устройствах. Основные протоколы:
  + **TCP (Transmission Control Protocol)** — надёжная доставка, контроль ошибок, порядок пакетов
  + **UDP (User Datagram Protocol)** — быстрая, но ненадёжная передача без подтверждений Этот уровень разбивает данные на сегменты и следит за их корректной доставкой.
* **Сетевой уровень** Отвечает за логическую адресацию и маршрутизацию пакетов. Основной протокол:
  + **IP (Internet Protocol)** — предоставляет IP-адресацию и доставку пакетов от узла к узлу. Здесь определяется путь, по которому пойдут пакеты через сеть.
* **Канальный уровень** Обеспечивает физическую передачу данных между двумя устройствами в пределах одного сегмента сети. Примеры технологий:
  + **Ethernet** — в проводных сетях
  + **Wi-Fi** — в беспроводных сетях Здесь данные упаковываются в кадры и передаются на физическом уровне.

**Маршрутизация** — это выбор маршрута, по которому сетевой пакет достигнет своей цели.

**Назначение маршрутизации:**

* Определять путь между источником и получателем;
* Доставлять пакеты к адресату через промежуточные устройства;
* Обеспечивать эффективное использование сетевых ресурсов.

**Классификация маршрутизации:**

* **Статическая** — маршруты прописываются вручную и не меняются автоматически;
* **Динамическая** — маршруты определяются автоматически на основе алгоритмов (например, RIP, OSPF, BGP);
* **Внутренняя маршрутизация (IGP)** — маршрутизация внутри одной автономной системы;
* **Внешняя маршрутизация (EGP)** — маршрутизация между автономными системами.

**Основные функции маршрутизаторов:**

* Выбор следующего узла (next hop) на маршруте;
* Обработка и пересылка пакетов;
* Адаптация к изменениям в топологии сети;
* Поддержка таблицы маршрутов и участие в протоколах обмена маршрутной информацией

**5. Назначение API**

**API (Application Programming Interface)** — это программный интерфейс, позволяющий разным программам взаимодействовать друг с другом.

**Назначение API:**

* **Инкапсуляция логики** — скрывает внутреннюю реализацию и предоставляет доступ только к нужным функциям;
* **Упрощение разработки** — программисты могут использовать готовые модули, не вникая в детали их реализации;
* **Стандартизация взаимодействия** — приложения и сервисы могут работать вместе, следуя единым соглашениям;
* **Масштабируемость и расширяемость** — API упрощает интеграцию новых компонентов в существующие системы.

Примеры:

* API веб-сервиса позволяет мобильному приложению запрашивать данные с сервера;
* API операционной системы предоставляет функции управления файлами или сетью;
* API библиотек и фреймворков помогает ускорить разработку без "изобретения велосипеда".

**6. Формы представления данных: JSON, XML**

Эти форматы используются для хранения и передачи структурированной информации между приложениями, особенно при работе с веб-сервисами.

**JSON (JavaScript Object Notation):**

* Основан на синтаксисе JavaScript;
* Представляет данные в виде пар "ключ-значение";
* Компактен, легко читается человеком;
* Поддерживается большинством языков программирования;
* Часто используется в REST API.

Пример:

{

  "name": "Alice",

  "age": 30

}

**XML (eXtensible Markup Language):**

* Основан на тегах, подобно HTML;
* Позволяет описывать вложенные иерархии;
* Более универсален, но громоздкий по синтаксису;
* Часто применяется в старых или сложных системах, а также в SOAP API.

Пример:

<person>

  <name>Alice</name>

  <age>30</age>

</person>

**Сравнение:**

* JSON проще, компактнее, быстрее парсится — поэтому он стал стандартом в веб-разработке;
* XML даёт больше возможностей для описания схем, валидации и расширения — его применяют там, где необходима строгость формата.

**7. Аутентификация и авторизация пользователей в клиент-серверных приложениях**

В клиент-серверной архитектуре важно контролировать, **кто** обращается к серверу и **что ему разрешено делать**. Для этого используются два ключевых механизма:

* **Аутентификация** Это процесс **подтверждения личности** пользователя. Цель — убедиться, что пользователь действительно тот, за кого себя выдаёт. Методы:
  + логин и пароль — самый распространённый способ;
  + одноразовые коды (например, 2FA через SMS или приложения);
  + токены доступа (например, JWT — JSON Web Token);
  + биометрические данные — отпечатки пальцев, распознавание лица;
  + внешние поставщики идентификации — OAuth (например, вход через Google, Yandex, VK и др.).
* После успешной аутентификации система знает, **кто** обращается к серверу.
* **Авторизация** Это процесс **предоставления доступа к ресурсам** на основе уже подтверждённой личности. Авторизация определяет:
  + какие действия разрешены пользователю;
  + к каким данным и функциям у него есть доступ;
  + что ему запрещено.
* Реализуется через:
  + списки прав и ролей;
  + политики доступа (например, в зависимости от контекста или условий);
  + ACL (Access Control Lists) и RBAC (Role-Based Access Control).

**Типичные схемы реализации:**

* **Сессионная модель** — сервер создаёт уникальную сессию и хранит её на своей стороне, клиент получает идентификатор (cookie);
* **Токен-ориентированная модель** — сервер выдаёт клиенту токен, который тот передаёт в каждом запросе (например, в заголовке Authorization). JWT-токены могут содержать зашифрованную информацию о пользователе;
* **OAuth2 / OpenID Connect** — используются для делегированной аутентификации и авторизации через сторонние сервисы.

**Итог:** Аутентификация отвечает на вопрос *«Кто ты?»*, а авторизация — *«Что тебе разрешено?»*.

**8. Масштабирование клиент-серверных приложений**

**Масштабирование** — это способность системы адаптироваться к увеличению нагрузки (например, рост числа пользователей, объёма данных или количества запросов).

Существует два основных подхода:

* **Горизонтальное масштабирование** Предполагает **добавление новых серверов** (нод, экземпляров приложения). Применяется в распределённых системах и облачных решениях. Примеры:
  + кластеризация серверов приложений;
  + балансировка нагрузки (load balancing) между несколькими машинами;
  + масштабирование баз данных (репликация, шардинг).
* Плюсы: высокая отказоустойчивость, масштабируемость без ограничений конкретного сервера. Минусы: усложнение архитектуры, необходимость синхронизации.
* **Вертикальное масштабирование** Это **увеличение ресурсов одного сервера** — процессора, оперативной памяти, хранилища. Пример: замена сервера на более мощный, установка большего объёма ОЗУ. Плюсы: простота реализации. Минусы: ограниченность ресурсов физической машины и точка отказа.

**Цель масштабирования**:

* обеспечить стабильную работу при росте нагрузки;
* избежать задержек и сбоев;
* сохранять доступность и производительность.

**Реальные подходы масштабирования включают:**

* автоматическое масштабирование в облаке (AWS Auto Scaling, Yandex Cloud, GCP);
* разделение функциональности (выделение микросервисов);
* оптимизацию запросов к базе данных и кэширование;
* использование CDN для разгрузки трафика.