**Вопросы к Зачету**

1. **ЧТО ТАКОЕ ТЕСТИРОВАНИЕ ?**

ТЕСТИРОВАНИЕ ПО – ЭТО ПРОВЕРКА СООТВЕТСТВИЯ МЕЖДУ РЕАЛЬНЫМ ПОВЕДЕНИЕМ ПРОГРАММЫ И ЕЕ ОЖИДАЕМЫМ ПОВЕДЕНИЕМ.

1. **ЦЕЛИ ТЕСТИРОВАНИЯ?**

* ПРЕДОСТАВЛЕНИЕ АКТУАЛЬНОЙ ИНФОРМАЦИИ О КАЧЕСТВЕ ПРОДУКТА И ЕГО СООТВЕТСТВИЯ ТРЕБОВАНИЯМ К НЕМУ ПРЕДЪЯВЛЯЕМЫМ
* НАХОЖДЕНИЕ БАГОВ ДО ТОГО, КАК ИХ НАЙДУТ ПОЛЬЗОВАТЕЛИ.
* ОЦЕНКА КАЧЕСТВА ДОРАБОТКИ;
* ПОДТВЕРЖДЕНИЕ СООТВЕТСТВИЯ ТРЕБОВАНИЯМ;
* ПРОВЕРКА РАЗНЫХ СЦЕНАРИЕВ ИСПОЛЬЗОВАНИЯ;
* ПРОВЕРКА БЕЗОПАСНОСТИ;

1. **ИЗ ЧЕГО СОСТОИТ РАБОТА ТЕСТИРОВЩИКА ?**

* **ИЗУЧЕНИЕ ТРЕБОВАНИЙ**
* **СОСТАВЛЕНИЕ ТЕСТОВОЙ ДОКУМЕНТАЦИИ(ЧЕК-ЛИСТЫ,ТЕСТ-КЕЙСЫ)**
* **ПРОВЕДЕНИЕ ТЕСТИРОВАНИЯ**
* **ОБНАРУЖЕНИЕ, ЛОКАЛИЗАЦИЯ, ОТСЛЕЖИВАНИЕ ДЕФЕКТОВ (BUG-РЕПОРТЫ, ОТЧЁТ О ТЕСТИРОВАНИИ)**
* **ВЗАИМОДЕЙСТВИЕ С КОМАНДОЙ РАЗРАБОТКИ И ЗАКАЗЧИКОМ**

1. **ЖИЗНЕННЫЙ ЦИКЛ ИЛИ ПРОЦЕСС РАЗРАБОТКИ ПО:**

* **Формирование требований**
* **Анализ требований**
* **Проектирование (дизайн)**
* **Кодирование**
* **Тестирование**
* **Внедрение**
* **Сопровождение и эксплуатация**
* **Готовый продукт**

1. **ЧЕМ ОТЛИЧАЕТСЯ QA ИНЖЕНЕР ОТ QC**

**QC-ТЕСТИРОВЩИКИ - ЗАНИМАЮТСЯ ТЕСТИРОВАНИЕМ ГОТОВОГО ПРОДУКТ, ПО ФАКТУ ПРОСТО ИЩУТ БАГИ, КОГДА УЖЕ РАЗРАБОТКА ЗАВЕРШЕНА, НЕ ПОДКЛЮЧАЮТСЯ НА САМЫХ РАННИХ ЭТАПАХ РАЗРАБОТКИ ПО   
QA-ИНЖЕНЕРЫ - ПОДКЛЮЧАЮТСЯ НА РАННИХ ЭТАПАХ РАЗРАБОТКИ, СЛЕДЯТ ЗА КАЧЕСТВОМ НА ЭТАПАХ СОСТАВЛЕНИЯ ТРЕБОВАНИЙ, ДИЗАЙНА, РАЗРАБОТКИ, ДУМАЮТ, КАК ПОМЕНЯТЬ ПРОЦЕССЫ, ЧТОБЫ ПРЕДОТВРАТИТЬ БАГИ В БУДУЩЕМ**

1. **ТЕСТОВЫЕ СТЕНДЫ**

**У вас было 3 стенда, на DEV- разрабы, STAGE - тестеры, PROD – пользователи**

1. **СТРАТЕГИИ РАЗРАБОТКИ**

* waterfall - следующий этап не может быть начат, пока не завершен предыдущий..
* **итерационная / спиральная или гибкая - строится в виде последовательности версий, каждая из которых дополняет предыдущую. разработка спринтами 2-4 нед, можно дополнять требовани**

1. **КАКОЙ ЗНАЕШЬ ГИБКИЙ ПОДХОД К РАЗРАБОТКЕ? AGILE**

* **Люди и взаимодействие важнее процессов и инструментов;  
  Работающий продукт важнее исчерпывающей документации;  
  Сотрудничество с заказчиком важнее согласования условий контракта;  
  Готовность к изменениям важнее следования первоначальному плану.**

1. **КАКИЕ ТРЕБОВАНИЯ БЫВАЮТ ?**

**Прямые** — это требования, которые явно описывают в задаче, что должна делать доработка. Они конкретны и измеримы.

**Косвенные — это требования, которые явно не описываются в задаче, но подразумеваются как само собой разумеющиеся**

1. **КРИТЕРИИ ХОРОШИХ ТРЕБОВАНИЙ**

* **полнота** все ли описано? ничего не забыли? вдруг у нас остался не описанный функционал?
* **однозначность** «отчет должен загружаться быстро» → что значит «быстро»?
* **непротиворечивость** написано, что любая страница должна грузится не более 3 секунд. аналитик пишет тз на новый модуль отчетности, который использует много данных, и там пишет, что страница может загружаться до 1 минуты.
* **тестируемость** бывает так, что разработчик уже всё сделал, и тут только тестировщик понимает, что  задачу никак нельзя проверить.

1. **ОСНОВНОЙ СОСТАВ КОМАНДЫ**

* **ПРОДЖЕКТ МЕНЕДЖЕР- руководитель проекта, ответственный за разработку и команду от планирования до реализации.**
* **ТИМЛИД- рулит командой разрабов и тестеров и помогает решать самые сложные задачи**
* **BACKEND-РАЗРАБОТЧИКИ - разрабатывает то, что под капотом, настривает бд, программирует процессы и выстраивает архитектуру взаимодействия функций**
* **FRONTEND-РАЗРАБОТЧИКИ- разрабатывают интерфейс. это всё, что видит пользователь, когда открывает сайт, а так же настравает связи с бэкендом**
* **АНАЛИТИК- связывает заказчика с командой разработки. формулирует требования заказчика и приводит их к виду технического задания, описывает требования в джире. чем предстоит заниматься.**
* **ПРОДУКТ ОВНЕР связывает команду и клиента**
* ДИЗАЙНЕР
* ТЕСТИРОВЩИКИ

1. **КТО ТАКОЙ ФУЛЛСТЕК РАЗРАБОТЧИК?**

Фуллстек разработчик — это специалист, который умеет работать как с фронтенд, так и с бэкенд частями веб-приложений. Это означает, что он разбирается во всем процессе создания приложения — от клиентской стороны (то, что видит и с чем взаимодействует пользователь) до серверной

1. **ВНУТРЕННЯЯ ТЕСТОВАЯ ДОКУМЕНТАЦИЯ ?**

* Набор тестовых артефактов (документов), используемых **внутри отдела тестирования** для повышения его эффективности
* **Чек-лист -(Список проверок, которые выполняет тестировщик чтобы оценить текущее состояние программного обеспечения или отдельной функциональности.)**
* **Тест-кейс -(Документ, содержащий последовательность действий направленных на проверку какого-либо функционала, описывающих как прийти к ожидаемому результат )**
* **Тест-комплект - Набор тест-кейсов, собранный по определенному принципу**
* **Тестовая стратегия стратегия — это высокоуровневый документ, описывающий общий подход к тестированию программного обеспечения в проекте. В нем излагаются цели тестирования, его процессы, виды тестирования, роли участников, инструменты, используемые для тестирования, а также критерии успеха и завершения. Тестовая стратегия помогает команде обеспечить качество продукта и избежать ошибок на всех этапах разработки.**
* **Тест-план — это документ, который детализирует конкретные действия по тестированию программного обеспечения для проекта.**

1. **Внешняя тестовая документация**

**Набор тестовых артефактов (документов), используемых как внутри отдела тестирования, так и за его пределами для повышения его эффективности тестирования и разработки**

* **Баг-репорт - Документ, описывающий ситуацию или последовательность действий, которая приводит к некорректной работе объекта тестирования.**
* **Запрос на изменение - Формальный запрос на внесение изменений в проект, требование, продукт или процесс. Оформляется, как правило, в виде таски.**
* **Отчет о тестировании - Документ, который мы заполняем после тестирования и описываем соответствие/ несоответствие продукта требованиям. Может быть простым комментарием со сводкой по тестированию.**

1. **БАГ-РЕПОРТ. ОБЯЗАТЕЛЬНЫЕ ПОЛЯ**

* **Заголовок Что? Где? Когда?**
* **Описание**
* **Шаги для воспроизведения**
* **Ожидаемый результат**
* **Фактический результат**
* **Приоритет**
* **Серьезность**
* **Исполнитель**
* **скриншот или скринкаст**

1. **Статусы Баг-репорта**

**Открыт - Начальное состояние бага. Он зафиксирован и заведен в систему.**

**В работе - Разработчик принял баг-репорт и начал исправление дефекта.**

**Исправлен (Готов к проверке) - Разработчик исправил дефект и вернул задачу на проверку.**

**Закрыт - Исправление проверено. Ошибка не повторяется.**

**Открыт повторно - Разработчик исправил дефект, вернул задачу на проверку, но ошибка не исправлена и возвращена на исправление.**

**Отклонен (Не удается воспроизвести) - Разработчик не смог повторить ошибку и выполнить исправление. Задача возвращена тестировщику для актуализации.**

**Отложен - Отложен до лучших времен :)**

1. **Серьезность - Уровень влияния ошибки на работоспособность приложения или функциональности. Опред. тестировщиком**

* **Блокирующая (Blocker)**. Ошибка, которая полностью блокирует или препятствует нормальной работе программного продукта. **Пример:** приложение для онлайн-банкинга не позволяет пользователям войти в свои аккаунты или осуществлять любые операции из-за ошибки в аутентификации или обработке транзакций. Это серьезно влияет на функциональность и удобство использования приложения, а также может привести к финансовым потерям или потере доверия со стороны клиентов.
* **Критическая (Critical) -** ошибка, которая может привести к серьезным проблемам или потенциальным угрозам безопасности. Пример: При переводе денег пользователю, деньги списываются со всех счетов сразу.
* **Значительная (Major) -** ошибка, которая может вызвать некритические проблемы или незначительные нарушения функциональности. Пример: При открытии истории переводов, отсутствует часть данных.
* **Незначительная (Minor) -** ошибка, которая имеет минимальное влияние на функциональность или пользовательский опыт.
* Пример: небольшое графическое искажение на веб-странице.
* **Тривиальная (Trivial)** - ошибка, не оказывающая практического влияния на работу программы или функциональность. Пример: опечатка в тексте Приоритет напрямую связан с бизнес-процессами продукта. Их хорошо знает и понимает только руководитель или PM. кнопки, которая не влияет на функциональность приложения.

1. **Что такое Приоритет?**

**Приоритет** напрямую связан с бизнес-процессами продукта. Их хорошо знает и понимает только руководитель или PM.

* **Высокий (High)** – исправить в первую очередь
* **Средний (Medium)** – исправить во вторую очередь, когда нет дефектов с высоким приоритетом
* **Низкий (Low)** – исправляется в последнюю очередь, когда все дефекты с более высоким приоритетом уже исправлены.

1. **ТЕХНИКИ ТЕСТ ДИЗАЙНА это этап процесса тестирования ПО, на котором проектируются и создаются тестовые случаи (тест кейсы).**

* **Эквивалентное разделение;** Разбиваем тестируемый объект на классы эквивалентности (допустимые значения и недопустимые).
* **Анализ граничных значений;** Методика тестирования ПО, которая проверяет поведение системы на границах значений входных данных.
* **Предугадывание ошибки;** Суть этой техники в том, что тестировщик предугадывает, где могут появиться ошибки, опираясь на свой опыт, знание системы и требования к продукту, и может уделить этим областям повышенное внимание.
* **Попарное тестирование;** Это комбинация пар значений так, чтобы каждый вводный параметр взаимодействовал с другими параметрами. Используется для сокращения тестовых проверок, при больших объемах вводных данных.
* **Техника тестирования пустых значений** Проверяет, как отреагирует система при вводе пустых значений в поле

1. **ПРИНЦИПЫ ТЕСТИРОВАНИЯ, НАС 7:**

* **ИСЧЕРПЫВАЮЩЕЕ ТЕСТИРОВАНИЕ НЕВОЗМОЖНО;** Насколько бы тщательным тестирование не было, нельзя учесть все возможные сценарии и предвидеть все возможные ошибки**.**
* **ТЕСТИРОВАНИЕ ДЕМОНСТРИРУЕТ НАЛИЧИЕ ДЕФЕКТОВ, А НЕ ИХ ОТСУТСТВИЕ;** Тестирование может выявить тот момент, что ошибки присутствуют, но не может доказать в полной мере, что дефектов нет.
* **ЗАБЛУЖДЕНИЕ ОБ ОТСУТСТВИИ ОШИБОК;** не существует какого-либо продукта без багов или ошибок.
* **РАННЕЕ ТЕСТИРОВАНИЕ СОХРАНЯЕТ ВРЕМЯ И ДЕНЬГИ;**
* **ПРИНЦИП СКОПЛЕНИЯ ИЛИ КЛАСТЕРИЗАЦИЯ ДЕФЕКТОВ; –** наибо́льшее количество дефектов 80% обычно содержится в небольшо́м количестве 20% модулей.
* **ТЕСТИРОВАНИЕ ЗАВИСИТ ОТ КОНТЕКСТА;** Для разного софта будут применяться разные подходы к его тестированию. К примеру, способ тестирования мобильного приложения будет отличаться от того, которым тестируется коммерческий сайт**.**
* **ПАРАДОКС ПЕСТИЦИДА.** Если к какому-либо функционалу применять постоянно повторяющийся набор тестов – то эти проверки в скором времени будут неэффективны в нахождении новых дефектов.

1. **ЧТО ТАКОЕ API?**

API –программный интерфейс, с помощью которого приложение(сайт) или один из его компонентов может взаимодействовать с другим приложением(сайтом) или компонентом как внешним, так и внутренним.

Самые популярные методы API:

GET – получить данные

POST – записать данные

PUT – заменить данные или создать

DELETE – удалить данные

+PATCH – обновить часть существующих данных

**CRUD —** создание (create) POST , чтение (read) GET , обновление (update) PUT и удаление (delete) DELETE - четыре основные метода для взаимодействия с ресурсами REST API. В REST-архитектуре CRUD соответствует следующим HTTP-методам.

**Метод OPTIONS: Метод OPTIONS используется для получения информации о том, какие методы HTTP поддерживаются сервером для конкретного ресурса. Он позволяет клиенту определить доступные действия, которые можно выполнить над ресурсом, не предпринимая самих действий.**

**Особенности:**

* **Возвращает заголовок Allow, содержащий список поддерживаемых методов.**
* **Часто используется в CORS (Cross-Origin Resource Sharing) для проверки разрешений на доступ к ресурсам с других доменов.**

**Метод HEAD: Метод HEAD запрашивает ресурс аналогично методу GET, но возвращает только заголовки HTTP-ответа без тела сообщения. Например, вы хотите узнать размер файла, доступного по URL, без загрузки самого файла.**

**Разница:**

* **OPTIONS возвращает информацию о доступных методах и разрешениях, а HEAD возвращает только заголовки, аналогичные GET-запросу, без содержимого.**
* **OPTIONS используется для проверки возможностей взаимодействия с ресурсом, а HEAD для получения метаинформации о ресурсе.**

1. **Коды ответов от API**

* 100 информационные
* 200 инф. Успешна
* 300 перенаправление
* 400 клиентская ошибка
* 500 ошибка сервера

101 Switching Protocols («переключение протоколов»);

102 Processing («идёт обработка»);

200 OK" указывает, что запрос выполнен успешно

201 Created. Это означает, что запрос был успешным и ресурс был создан. Код используется для подтверждения успеха запроса PUT или POST

204 No Content («нет содержимого»)

300 Moved Permanently. Этот код ответа означает, что URL-адрес запрошенного ресурса был изменен и произошел редирект.

400 Bad Request. Запрос был сформирован неверно. Это происходит с запросами POST и PUT, когда данные не проходят проверку или имеют неправильный формат.

401 Unauthorized. Эта ошибка указывает на то, что вам необходимо выполнить аутентификацию перед доступом к ресурсу.

403 Forbidden — что доступ к запрошенному ресурсу запрещен.

404 Not Found. Этот код показывает, что не удалось найти требуемый ресурс. 404 означает, что URL-адрес не распознается или запрашиваемого ресурса нет в указанном месте.

409 Conflict. Код указывает на произошедший конфликт. Например, вы используете запрос PUT для создания одного и того же ресурса дважды.

500 Internal Server Error. Как правило, ответ 500 используется, когда обработка запроса завершается неудачно из-за непредвиденных обстоятельств на стороне сервера.

1. **Какие типы данных можно отправить и получить?**

**String(Стринг) – строка “Миша” { "name":"John" }**

**Integer(Интеджер) – число 123 { "age":30 }**

**Float(Флоат) – число с плавающей точкой 12.3 { "age":30.5 }**

**Массив – коллекция значений одного типа [12,13,14,84,154] { "employees":[ "John", "Anna", "Peter" ] }**

**Объект – коллекция пар ключ-значение { "employee":{ "name":"John", "age":30, "city":"New York" } }**

**Boolean(Булеан) – True; False. { "sale":true }**

**Null (Нул)– ноль (отдельный тип данных - пустота)**

1. **Обмен данными**

**JSON (англ. *JavaScript Object Notation*) — текстовый формат обмена данными в *REST API*.  
  
XML — для отображения, хранения и транспортировки информации.   
В XML вы изобретаете свои собственные теги, так как в языке нет предопределенных тегов. В*SOAP API* возможен только формат *XML*, а вот *REST API* поддерживает как *XML*, так и *JSON*.   
  
Разработчики предпочитают *JSON*— он легче читается человеком и меньше весит.**

1. **SWAGGER**

**Swagger — это набор инструментов для создания и описания API, что упрощает взаимодействие между разработчиками фронтенда и бекенда, а также тестирование API.**

**Основные компоненты Swagger включают в себя:**

**OpenAPI Specification : Это формат описания структуры API, позволяя документировать каждый эндпоинт, параметры запроса и ответы.**

**Swagger Editor: Интерактивный веб-интерфейс, который позволяет разработчикам создавать, редактировать и визуализировать файлы спецификации OpenAPI.**

**Swagger UI: Готовый к использованию интерфейс пользователя для визуализации и тестирования API. Позволяет взаимодействовать с API прямо из браузера.**

**SWAGGER тоже нужно проверять**

1. **Модель запроса**
2. **Модель ответа 200**
3. **Модель ответа 400/500**
4. **Домен и протокол**
5. **Обязательность полей**
6. **Postman**

Это [HTTP-клиент](https://www.postman.com/) для тестирования API. HTTP-клиенты тестируют отправку запросов с клиента на сервер и получение ответа от сервера.

Для работы с серверами программа использует протокол HTTP. Тестировщик отправляет тестовые запросы от клиента на сервер и получает ответ, есть ли ошибка в работе API.

Postman collection — это файл, созданный в Postman, который содержит набор HTTP запросов, ответов и сопутствующей информации, связанной с тестированием API. Коллекция в Postman обычно включает в себя следующую информацию: HTTP запросы: GET, POST, PUT, DELETE и другие типы запросов.

Авторизация

* API key – в заголовок или в параметры добавляется пара ключ-значение, его выдают на проекте коллеги или есть в доке
* Bearer token – токен на предъявителя, это веб-маркер JSON, представляет собой набор символов, передается в заголовок запроса.

Basic auth — просто передача при запросе “Логин/пароля” от системы, например от Jira

Если у API есть авторизация, то при проверке добавляется 2 кейса  
+ Ключ передан - все ок, апи работает  
- Ключ не передан/передан не в том формате/просрочен/заблокирован - система выдает понятную ошибку и 401 код

1. **HTTP-заголовки**

В HTTP-запросах заголовки — это поля, предоставляющие дополнительную информацию от клиента серверу и наоборот.

Если вы видите, что у API в доке есть заголовки, которые имеют функциональность для пользователя, язык например, то надо проверять корректную работу этих заголовков   
(Позитивные проверки и негативные на каждый важный заголовок)

1. Идемпотентность

Идемпотентность в контексте API означает, что повторное выполнение одного и того же запроса не приведет к изменению состояния системы. Если запрос успешно выполнен один раз, повторное его выполнение не приведет к дополнительным изменениям или побочным эффектам.  
  
POST - не идемподентный, тк он каждый раз создает новую запись (если не сделано блока в API на повторную запись с теми же данными)

1. **Общие правила тестирования API**

Изучаем документацию, модель запроса, модели ответа, заголовки, обязательность передачи параметров, коды ответа и структуру JSON, наличие авторизации, срок жизни некоторых параметров, уточняем у разрабов, если что-то не понятно

Бизнес-логика: составляем чек лист проверок, разделяя их на позитивные и негативные.  
  
Обращаем внимание на статус код, чтобы в зависимости от запроса он был верный (200е/400е и тд)  
  
Сверяем структуру ответа с документацией в разных сценариях (позитивные, негативные)

1. **Идентификация, аутентификация, авторизация**

**Идентификация:** Уникальное указание кто-то или что-то.

Аутентификация: Подтверждение личности на основе предоставленных учетных данных.

**Авторизация:** Предоставление прав доступа после успешной аутентификации.

Пример: Представьте, что вы идентифицируетесь по своему имени пользователя, аутентифицируетесь с использованием пароля, и после успешной аутентификации вам предоставляются определенные права доступа (авторизация) для работы с ресурсами в системе.

1. **Клиент-серверное взаимодействие**

**Клиент — любой инструмент, который действует от имени пользователя.**

**Клиент всегда инициирует запрос.**

**В основном эту роль выполняет веб-браузер, в некторых случаях сервер инициирует запрос к другому серверу**

**Сервер — находится на другой стороне канала связи, который обрабатывает запросы клиента.   
  
Для пользователя сервер выглядит как одна виртуальная машина, на самом деле это может быть набор серверов, разделяющих нагрузку.**

1. **Из чего состоит клиент серверная Архитектура?**

**Клиент -Примеры: Веб-браузеры, мобильные приложения.**

**Сервер -Примеры: Веб-серверы, серверы баз данных.**

**Сетевое Соединение -Примеры: Интернет, локальные сети (LAN).**

**Протоколы- Примеры: HTTP/HTTPS, FTP, SMTP.**

**Запросы и Ответы- Примеры: GET запрос, POST запрос.**

**Аутентификация и Авторизация -Примеры: Логин и пароль, токены аутентификации.**

**Кэширование - Примеры: Кэш браузера, серверное кэширование.**

1. **Что такое DNS сервер?**

DNS (Domain Name System) сервер — это сервер, который отвечает за преобразование доменных имен в IP-адреса и наоборот. Проще говоря, DNS позволяет людям использовать понятные имена веб-сайтов (например, example.com) вместо сложных для запоминания числовых IP-адресов (например, 93.184.216.34).

Как работает DNS:

Запрос доменного имени: Когда пользователь вводит в браузере адрес сайта (например, example.com), компьютер отправляет запрос на DNS-сервер, чтобы получить соответствующий IP-адрес.

Поиск IP-адреса: DNS-сервер ищет IP-адрес, который сопоставлен с этим доменным именем.

Если DNS-сервер знает IP-адрес (он сохранен в его кэше), он сразу возвращает его.

Если нет, сервер передает запрос другим DNS-серверам до тех пор, пока не будет найден нужный IP-адрес.

Возвращение IP-адреса: Когда IP-адрес найден, DNS-сервер возвращает его компьютеру пользователя.

Подключение к сайту: Браузер использует полученный IP-адрес для подключения к серверу, на котором размещен веб-сайт.

Зачем нужен DNS:

Упрощение: Людям легче запомнить доменные имена, такие как example.com, чем числовые IP-адреса.

Универсальность: Веб-сайты могут менять свои IP-адреса, но пользователи по-прежнему смогут заходить на сайт, используя одно и то же доменное имя.

Оптимизация запросов: DNS-серверы могут кэшировать результаты запросов, что позволяет ускорить доступ к часто используемым сайтам.

Таким образом, DNS играет ключевую роль в том, как работает интернет, облегчая пользователям навигацию по сайтам и обеспечивая эффективное взаимодействие между компьютерами.

1. **Что такое прокси?**

Прокси-сервер (или просто прокси) — это посредник между устройством пользователя (например, компьютером) и интернетом. Когда вы используете прокси, ваши запросы к веб-сайтам или другим интернет-ресурсам проходят через этот сервер, который отсылает их от вашего имени. Ответы от ресурсов также поступают сначала на прокси, а уже затем передаются вам.

Как работает прокси:

Пользователь отправляет запрос на доступ к веб-странице (например, example.com).

Прокси-сервер получает этот запрос, перенаправляет его на веб-сайт и ждет ответа.

Веб-сайт отвечает прокси с нужной информацией.

Прокси передает ответ (например, веб-страницу) пользователю.

Основные задачи и преимущества прокси:

Анонимность: Прокси может скрыть ваш настоящий IP-адрес, показывая веб-сайтам IP-адрес прокси-сервера вместо вашего. Это повышает вашу конфиденциальность в интернете.

Доступ к заблокированным ресурсам: Если сайт или сервис заблокирован в вашей стране или сети, вы можете использовать прокси-сервер, расположенный в другом регионе, чтобы обойти эти ограничения.

Кэширование: Некоторые прокси-серверы могут сохранять копии веб-страниц. Это ускоряет доступ к часто посещаемым ресурсам, так как сервер может выдать кэшированную версию страницы вместо запроса на исходный сайт.

Контроль доступа и фильтрация контента: Прокси часто используются в корпоративных сетях для контроля доступа к интернету, фильтрации контента (например, блокировки определенных сайтов) и мониторинга активности пользователей.

Оптимизация трафика: Прокси может сжимать данные или блокировать нежелательный контент (например, рекламу), уменьшая объем передаваемой информации и ускоряя загрузку.

**Виды прокси-серверов:**

HTTP-прокси: Работает только с веб-трафиком (HTTP и HTTPS). Обычно используется для анонимного серфинга в интернете.

SOCKS-прокси: Работает на более низком уровне сети и поддерживает любые протоколы, включая email, FTP, видео-стриминг и т.д. Это более универсальный тип прокси.

Транспарентный прокси: Не скрывает IP-адрес пользователя и часто используется для кэширования данных и фильтрации трафика на корпоративных сетях или в образовательных учреждениях.

1. **Двухзвенная архитектура**

**Двухуровневая(двухзвенная) клиент-серверная архитектура — это, когда у нас есть клиент и сервер с базой данных.**

**Данные хранятся на сервере (второе звено) и предоставляются клиенту (первое звено) в ответ на запросы.   
  
Сервер выполняет две основные роли: предоставляет данные и обрабатывает запросы от клиента.**

1. **Трехзвенная архитектура**

**Трехзвенная клиент-серверная архитектура включает три основных элемента:   
клиент, сервер приложений и сервер базы данных.**

**Клиент отвечает за отображение информации пользователю,   
  
Сервер приложений обрабатывает логику работы приложения,   
  
Сервер базы данных хранит и управляет данными. Такая архитектура помогает лучше организовать систему, разделяя ответственность между компонентами и обеспечивая более эффективное управление данными и бизнес-логикой.**

1. **Что такое REST архитектура ?**

**REST - это архитектурный стиль, предложенный в 2000 году. Он представляет собой набор ограничений и принципов для построения масштабируемых и гибких веб-сервисов.**

**Принципы   
Использование ресурсов, каждый должен быть отдельным (Resources):**

**Ограниченный интерфейс, ничего лишнего**

**Представление ресурсов в формате JSON, XML или HTML**

**Единообразие - все запросы и ответы в одном формате**

**Статическое управление состоянием. Каждый запрос от клиента к серверу должен содержать всю необходимую информацию  
Возможность кэширования**

**Клиент-серверная архитектура**

1. **Кеширование**

Кеширование в REST-архитектуре позволяет улучшить производительность и снизить нагрузку на сервер, сохраняя копии ранее полученных ответов на запросы.   
Важно учитывать, что данные в кэше могут быть устаревшими, поэтому необходимо правильно управлять его сроком жизни.

* **Кешируются методы, которые получают информацию**
* **GET запросы**
* **HEAD запросы**
* **OPTIONS запросы**

**Другие запросы, которые не изменяют состояние сервера и не содержат конфиденциальной информации, также могут быть кешированы в соответствии с правилами кеширования, определенными в заголовках ответа сервера.**

1. **Кеширование**

**Назначение: Кэш используется для хранения временных данных с целью ускорения доступа к ресурсам и уменьшения нагрузки на сервер.**

**Местоположение: Кэш может находиться на клиенте (в браузере) или на сервере (серверный кэш).**

**Содержимое: В кэше хранятся копии статических ресурсов (например, изображения, CSS, JavaScript) и динамических данных.**

**Срок действия: Кэш имеет срок действия, который устанавливается через заголовки HTTP (например, Cache-Control, Expires).**

**Контроль: Управляется через заголовки HTTP и политики кэширования на стороне сервера и клиента.**

1. **Куки**

**Назначение: Куки используются для хранения небольших фрагментов данных на стороне клиента, которые отправляются на сервер с каждым запросом. Это полезно для аутентификации, сессий, персонализации и отслеживания.**

**Местоположение: Куки хранятся на клиенте (в браузере) и отправляются на сервер с каждым HTTP-запросом к соответствующему домену.**

**Содержимое: В куки хранятся пары ключ-значение, такие как идентификаторы сессий, настройки пользователя, данные авторизации.**

**Срок действия: Куки имеют срок действия, который может быть установлен через параметры куки (например, Expires, Max-Age).**

**Контроль: Управляется через заголовки HTTP (Set-Cookie) и через JavaScript на стороне клиента.**

**Кэш хранит ресурсы для ускорения загрузки, а куки хранят данные для сессий и персонализации.**

**Кэш управляется через заголовки кэширования, а куки через заголовки Set-Cookie и JavaScript.**

**Кэш может быть как на стороне клиента, так и на сервере, а куки хранятся только на стороне клиента.**

1. **А как выглядит не RESTful API**

**Использование не всех методов HTTP: Может поддерживаться только ограниченный набор методов, например, только GET и POST, без PUT и DELETE.**

**Отсутствие четкого разделения ресурсов: Вместо четко определенных конечных точек для каждого ресурса, может использоваться одна конечная точка с различными параметрами.**

**Отсутствие самодокументируемости: В полном REST API ресурсы должны содержать достаточно информации для взаимодействия с ними. В неполном API может отсутствовать достаточная информация для клиента.**

**Неполные или нестандартные коды состояния HTTP: Использование только одного или двух кодов состояния, например, всегда 200 OK, даже при ошибках.**

**Отсутствие использования заголовков: Могут не использоваться важные заголовки HTTP, такие как Content-Type или Authorization.**

**Структура URL может быть непоследовательной: Вместо логичной иерархической структуры URL, могут использоваться непоследовательные или сложные URL.**

**Необработанные ошибки: В случае ошибок сервер может не возвращать детализированные сообщения об ошибках.**

1. **REST и SOAP**

**Есть два архитектурных подхода по реализации WEB сайтов - это SOAP и REST, отличие в том что:**

**REST - архитектурный подход к реализации веб сайтов на основе НТТР протокола.**

**RESTful- используют такие стандарты, как HTTP, URL, JSON и, реже, XML.**

**SOAP – это семейство протоколов и стандартов, это более тяжеловесный и сложный вариант с точки зрения машинной обработки.**

**В SOUP данные передаются в формате XML, а в REST в формате JSON.**

**SOAP используют HTTP как транспортный протокол, в то время как REST базируется на нем.**

**Поэтому REST работает быстрее и 90% сайтов строят на нём.**

**SOAP пример Представьте что у вас есть некий сервер, с которым вам надо общаться. Например веб-сервер.**

**Этот сервер может выполнять множество действий, работать с базой, выполнять какие-то сторонние запросы к другим серверам, заниматься каким-то вычислениями и т.д.**

**С таким сервером общаться человеку неинтересно, потому что он может не уметь/не хотеть отдавать красивые странички с картинками и прочим юзер-френдли контентом. Другие системы, обращаясь к этому серверу уже могут распоряжаться полученными от этого сервера данными по своему усмотрению - обрабатывать, накапливать, выдавать своим клиентам и т.д.**

**Ну вот, один из вариантов общения с такими серверами - это SOAP. SOAP протокол обмена xml-сообщениями.**

1. **Тонкий и толстый клиент**

**Толстый клиент:**

**Пример: Программа на твоем компьютере, которая сама обрабатывает много процессов и хранит часть инфы на компе, например, игровое приложение, которое установлено у тебя на компьютере или почта.**

**Тонкий клиент:**

**Пример: Веб-браузер, через который ты работаешь в онлайн-почте или гугл-доках. Он использует сервер, чтобы обрабатывать и хранить данные, а сам браузер просто показывает тебе результат.**

**Преимущества толстых клиентов:**

1. **Большая функциональность;**
2. **Наличие многопользовательского режима;**
3. **Возможность работы в режиме оффлайн**

**Недочеты:**

1. **Ресурсы**
2. **Обновления**
3. **Лицензии**
4. **Обновления данных**

**Плюсы тонкого клиента:**

1. **Минимальное аппаратное обслуживание;**
2. **Низкий риск возникновения неисправности;**
3. **Минимальные технические требования к аппаратному оборудованию.**

**Недочеты:**

1. **При сбое на сервере «пострадают» все подключенные пользователи;**
2. **Нет возможности работать без активного подключения к сети;**
3. **HTTP протокол**

**— это набор соглашений, которые определяют обмен данными между различными программами. Эти соглашения задают единообразный способ передачи сообщений и обработки ошибок.**

**Основные характеристики HTTP:**

**Клиент-серверная модель:**

**Клиент (например, браузер) отправляет запрос к серверу.**

**Сервер обрабатывает запрос и возвращает ответ (например, веб-страницу или другой ресурс).**

**Простота: HTTP — текстовый протокол. Запросы и ответы представляют собой текстовые сообщения, которые легко читаются и понимаются как компьютерами, так и людьми.**

**Статус соединений: HTTP — это статический протокол (stateless), что означает, что каждое соединение между клиентом и сервером является независимым. Сервер не сохраняет информацию о предыдущих запросах клиента. Для сохранения состояния сессий часто используются такие технологии, как cookie.**

**Из чего состоит HTTP/S протокол?**

* **HTTP-**[**метод**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods)**. Обычно, клиент хочет получить ресурс (используя GET) или передать значения HTML-формы (используя POST):**
* **Путь к ресурсу: URL;**
* **Версия HTTP-протокола**
* **Заголовки, он передает основную информацию о запросе и содержит пары ключ-значение;**
* **Тело запроса – опционально (для POST, PUT, DELETE, PATCH).**

1. **Из чего состоит URL адрес?**

**Протокол, по которому передаются данные(http, https)**

**Домен – адрес сайта в интернете**

**Путь – откуда именно берутся данные**

**Строка запроса (состоит из параметров)**

1. **Параметры URL**

Если разбить адрес на части, то сначала идёт протокол – https:// (англ. HyperText Transfer Protocol Secure) и за ним следует доменное имя страницы, в нашем случае – это www.onlinestore.com.

Теперь, мы добавим параметр, который состоит из переменной и значения. Выглядит это так: *?parameter=value*

Знак вопроса «?» это символ, который указывает на начало запроса. Ставится он в конце исходной ссылки. Знак равенства «=» отделяет переменную от значения и образует пару.

Таким образом, мы получаем следующий URL-адрес: *https://www.onlinestore.com?parameter=value*

Если вы хотите указать не одну, а несколько пар параметров, используйте символ амперсанда «&», чтобы отделить последующие пары значений. Пример: *https://www.onlinestor.com?product=socks&color=black&size=m*

В случае работы с трекером, вы можете добавлять динамические параметры в URL-ссылку. Их вписывают немного другим способом, а именно: *https://www.onlinestore.com?value={parameter}*

Переменная в фигурных скобках *{ }* сообщает системе, что вы хотите получить обратно данные этого значения после того как пользователь кликнул на рекламу. Так, трекер получает данные с источника трафика и записывает их в статистику.

1. **DevTools (или инструменты разработчика)**

**DevTools (или инструменты разработчика) — это набор встроенных инструментов, доступных в современных браузерах, таких как Chrome, Firefox, Safari и других. Они позволяют разработчикам анализировать и отлаживать веб-страницы прямо в браузере.**

**Рассмотрим основные функции DevTools на примере Chrome DevTools:**

**1. Elements (Элементы)**

* **Исследование и изменение DOM: Можно просматривать и редактировать структуру HTML-документа, изменять атрибуты элементов, добавлять или удалять элементы. (проверяли как добавляются параметры, как прокидывается что находится в хеддере)**
* **Стили и CSS: Отображает примененные к элементу стили, где можно изменить CSS-свойства, чтобы увидеть, как они повлияют на внешний вид страницы. (меняли цвет размер кнопок, заголовка и пр.)**
* **Box Model: Показывает модель коробки (Box Model) для выбранного элемента, включая размеры, отступы (margin), поля (padding) и рамки (border).**

**2. Console (Консоль)**

* **Выполнение JavaScript-кода: Позволяет выполнять JavaScript-код прямо в браузере, что удобно для тестирования и отладки.**
* **Логирование: Можно просматривать сообщения отладки, ошибки и предупреждения, которые были выведены с помощью console.log(), console.error(), console.warn() и других методов.**
* **Работа с объектами и функциями: Позволяет исследовать и манипулировать объектами и функциями, доступными на странице. (проверяли какие ошибки выпадают и насколько они критичны для работы. И проверяли как параметры прокидываются в яндекс метрику)**

**3. Sources (Источники)**

* **Просмотр и редактирование исходного кода: Можно просматривать и редактировать загруженные файлы JavaScript и CSS. (проверка ошибок в файлах-индексов. VK ошибка ссылки например. )**
* **Точки останова (breakpoints): Можно ставить точки останова в коде, чтобы приостановить выполнение скриптов и исследовать текущее состояние программы.**
* **Call Stack и Scope: Показывает стек вызовов и области видимости переменных, что помогает при отладке.**

**4. Network (Сеть)**

* **Отслеживание сетевых запросов: Позволяет отслеживать все сетевые запросы, включая файлы, загружаемые с сервера (HTML, CSS, JavaScript, изображения и т. д.). (проверка запросов ответов как frontend закидывает данные в APIшку, проверяли соответствие параметров Используя PRESERVE LOG- сохранение журнала DISABLE CASHE- очистить кэш)**
* **Информация о запросах и ответах: Отображает заголовки, статус-коды, время загрузки, размер и типы загружаемых файлов.**
* **Throttling: Можно симулировать разные скорости интернет-соединения (например, 3G), чтобы проверить, как будет вести себя приложение при медленном интернете.**

**5. Performance (Производительность)**

* **Анализ производительности: Позволяет записывать и анализировать производительность веб-страницы, включая время загрузки, рендеринга и выполнения скриптов.**
* **Фреймы и события: Отображает кадры (frames) и события, такие как события клика или скроллинга, чтобы можно было понять, что влияет на производительность.**

**6. Memory (Память)**

* **Профилирование памяти: Позволяет анализировать использование памяти веб-страницей и искать утечки памяти.**
* **Heap Snapshot: Снимок кучи (heap snapshot) помогает увидеть, какие объекты занимают память, и можно ли их удалить.**

**7. Application (Приложение)**

* **Работа с хранилищами: Позволяет просматривать и редактировать содержимое локального хранилища (Local Storage), сессионного хранилища (Session Storage) и IndexedDB.**
* **Cookies: Отображает все куки, установленные веб-сайтом, включая их значение, время истечения и доступные пути. Работа с куки как добавляются и удаляются. (Например до 1 минуты сокращали хранение куки и смотрели удалится ли. )**
* **Service Workers и Cache: Позволяет управлять сервис-воркерами и кэшем веб-приложения (Application Cache, Cache Storage).**

**8. Security (Безопасность)**

* **Сведения о сертификате: Показывает информацию о SSL/TLS сертификате, используемом для защищенного соединения.**
* **Мелкие уязвимости: Указывает на потенциальные проблемы безопасности, такие как использование устаревших протоколов или уязвимых шрифтов.**

**9. Lighthouse**

* **Аудит производительности: Проводит автоматический аудит веб-сайта и предоставляет рекомендации по улучшению производительности, доступности, SEO и других метрик.**

**10. Device Mode (Режим устройств)**

* **Эмуляция устройств: Позволяет эмулировать различные устройства, разрешения экрана и поведение сенсорного экрана для тестирования адаптивности и отзывчивости страницы.**

**Эти инструменты помогают разработчикам быстро находить и устранять ошибки, оптимизировать производительность и улучшать пользовательский интерфейс веб-сайтов.**

1. **/**
2. **/**
3. **/**
4. **/**
5. **/**
6. **/**
7. **/**
8. **/**
9. **/**
10. **/**
11. **ГИТЛАБ**

GitLab — это платформа для разработки, интеграции и развертывания программного обеспечения, которая объединяет функционал системы контроля версий (Git), непрерывной интеграции/непрерывного развертывания (CI/CD), управления проектами и многое другое. Одной из ключевых функций GitLab является возможность автоматизации процессов деплоя (развертывания) приложений.

**Основные понятия и компоненты для деплоя в GitLab**

**GitLab CI/CD (Continuous Integration/Continuous Deployment)**

**CI (Непрерывная интеграция):** Процесс автоматической сборки и тестирования кода каждый раз, когда изменения вносятся в репозиторий. Это помогает обнаружить ошибки на раннем этапе.

**CD (Непрерывное развертывание):** Автоматизация процесса развертывания приложения в различные среды (например, staging, production) после успешного выполнения всех тестов.

.**gitlab-ci.yml**

Это конфигурационный файл, расположенный в корневом каталоге репозитория, который определяет все этапы (stages) и задачи (jobs) для CI/CD пайплайнов.

Он позволяет указать, какие задачи выполнять, в каком порядке и на каких средах (environments). Например, можно определить стадии сборки, тестирования и деплоя.

Pipeline

**Пайплайн** — это последовательность этапов и задач, которые выполняются последовательно или параллельно для автоматизации различных процессов: сборки, тестирования, деплоя и т. д.

Пайплайн запускается автоматически при каждом изменении кода (например, при коммите или слиянии веток).

Runners

**GitLab Runner** — это агент, который выполняет задачи (jobs), определенные в .gitlab-ci.yml.

Runner может быть установлен локально, на сервере или в облаке, и поддерживает разные среды выполнения (например, Docker, Shell, SSH).

**Environments** (Среды)

Environments позволяют определять различные среды развертывания, такие как development, staging, production.Они помогают отслеживать, какое приложение и какой версии развернуто на конкретной среде.Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

1. **Что такое репозиторий ?**

Репозиторий — это хранилище, в котором хранится исходный код проекта, а также вся история изменений, метаданные и другие файлы, необходимые для разработки. Репозиторий может быть локальным (на вашем компьютере) или удалённым (на сервере, таком как GitHub, GitLab, Bitbucket и т.д.). Репозитории используются для организации и управления версиями кода, что особенно полезно при командной разработке.

**репозиторий   
мастер,** главная или основная [ветка](https://github.com/web-standards-ru/dictionary/blob/main/git.md#branch) репозитория.  
**мерж,** слияние изменений из одной ветки в другую ветку того же репозитория.  
**коммит,** фиксация изменений и запись их в репозиторий.

**ревью кода,** процесс проверки кода на соответствие требованиям, задачам и внешнему виду.  
**ветка,** параллельная версия репозитория. Она включена в этот репозиторий, но не влияет на другие версии, тем самым позволяя разработчикам независимо работать с кодом одного проекта.

**пуш,** отправка всех неотправленных коммитов на [удалённый репозиторий](https://github.com/web-standards-ru/dictionary/blob/main/git.md#repository).  
**репозиторий,** папка, в которой находятся: конфиги, журналы операций, индекс файлов и сами файлы репозитория.  
**локальный репозиторий,** репозиторий на локальном компьютере разработчика, где происходит разработка и фиксируются изменения, которые отправляются на удалённый репозиторий.

**удалённый репозиторий,** копия репозитория, хранящаяся отдельно от локального (обычно на удалённом сервере, например [Гитхаб](https://github.com/web-standards-ru/dictionary/blob/main/git.md" \l "GitHub)).

CI/CD или CICD — это комбинация непрерывной интеграции (continuous integration) и непрерывного развертывания (continuous delivery или continuous deployment) программного обеспечения в процессе разработки. CI/CD объединяет разработку, тестирование и развёртывание приложений.

Изображение выглядит как текст, снимок экрана, диаграмма, круг

Автоматически созданное описание

1. **SQL**

**SQL (Structured Query Language) - это декларативный язык программирования, используемый для управления, создания и модификации реляционных баз данных, состоящий из набора операторов и функций на английском языке.**

**SQL позволяет создавать, изменять и удалять данные в базе данных, а также извлекать информацию из неё для выполнения аналитических исследований.**

**SQL (Structured Query Language) —** это язык программирования, предназначенный для управления и манипулирования данными в реляционных базах данных. SQL используется для создания, модификации, извлечения и удаления данных, а также для управления доступом к данным и их структурой. SQL является стандартом для работы с большинством реляционных СУБД (систем управления базами данных), таких как MySQL, PostgreSQL, Microsoft SQL Server, SQLite и Oracle.

**Создание тестовых данных:**

Тестировщик может создать тестовую учетку для проверки удаления пользователя.

Или поставить 0 в количестве товара, чтобы проверить как фронт покажет плашку “распродано” на пустой остаток.

**Проверка целостности данных:**

Проверить, что все числа, например, в столбце "Возраст" передаются положительными значениями, а не Null или что-то еще.

**Тестирование функциональности приложения:**

Проверить, что приложение корректно сохраняет информацию о новых пользователях в базе данных при регистрации или добавлении товара в корзину

**Тестирование интеграции с другими системами:**

Проверить, что при оформлении заказа уходит запрос к базе данных и нужные параметры сохраняются в бд. А на основе них происходит отправка СМС например

**Тестирование сценариев использования:**

Тестировщик проверяет, что приложение правильно обрабатывает различные сценарии использования, такие как добавление и удаление товаров из корзины покупок.

1. **Базы данных (БД) —** это организованные структуры для хранения, управления и обработки данных. Существует несколько типов баз данных, каждая из которых предназначена для решения определенных задач и подходит для различных типов приложений. Рассмотрим основные виды баз данных и их особенности.

**Ключевое отличие реляционных от нереляционных БД заключается в способе организации данных: реляционные базы данных используют таблицы и отношения, в то время как нереляционные базы данных могут использовать различные модели хранения данных и не обязательно следуют строгой схеме. Реляционной база называется потому что данные в ней хранятся в взаимосвязанных между собой таблицах.**

**1. Реляционные базы данных (RDBMS) SQL**

Реляционные базы данных организуют данные в виде таблиц, где строки представляют записи, а столбцы — атрибуты данных. Взаимосвязи между таблицами устанавливаются с помощью внешних ключей. Основной язык для работы с реляционными базами данных — SQL (Structured Query Language).

Примеры: MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database, SQLite.

Особенности:

Строгая структура данных, основанная на таблицах.

Поддержка транзакций и целостности данных.

Использование языка SQL для работы с данными.

Хорошо подходят для систем, где важны точность и согласованность данных (финансовые системы, системы управления запасами и т.д.).

NO QSL

2. **Документоориентированные базы данных**

Документоориентированные базы данных хранят данные в виде документов, обычно в формате JSON, BSON или XML. Каждый документ содержит данные и их структуру, что позволяет хранить разнородные данные в одной коллекции.

**Документо-ориентированная база данных: Этот тип баз данных хранит данные в виде документов, обычно в формате JSON или BSON. Каждый документ содержит пары ключ-значение и может иметь различную структуру.**

**Примером такой базы данных является MongoDB**

Примеры: MongoDB, CouchDB, RavenDB.

Особенности:

Гибкая схема данных, позволяющая хранить различные типы документов в одной коллекции.

Поддержка сложных вложенных структур.

Хорошо подходят для работы с полуструктурированными данными, веб-приложениями, хранения информации о пользователях и контента.

3. **Колонкоориентированные базы данных**

Колонкоориентированные базы данных хранят данные по столбцам, а не по строкам, что позволяет эффективно обрабатывать большие объемы данных и выполнять аналитические запросы.

**Столбцовая (колоночная) база данных: В этом типе баз данных данные хранятся в виде столбцов, что позволяет эффективно работать с большими объемами данных и производить аналитические запросы.**

**Примером столбцовой базы данных является Apache Cassandra.**

Примеры: Apache Cassandra, HBase, Amazon Redshift, Google Bigtable.

Особенности:

Высокая производительность для операций агрегации и анализа.

Эффективное сжатие данных и быстрый доступ к столбцам.

Хорошо подходят для аналитических задач, систем хранения данных (data warehousing) и обработки больших данных (Big Data).

4. **Графовые базы данных**

Графовые базы данных хранят данные в виде узлов (вершин) и рёбер, которые представляют отношения между узлами. Они позволяют эффективно работать с данными, имеющими сложные взаимосвязи.

**Графовая база данных: В этом типе баз данных данные представлены в виде узлов (вершин) и связей (ребер), что позволяет эффективно моделировать и обрабатывать сложные сетевые структуры.**

**Примером графовой базы данных является Neo4j.**

Примеры: Neo4j, Amazon Neptune, ArangoDB, OrientDB.

Особенности:

Поддержка сложных связей между объектами, которые трудно моделировать в реляционных БД.

Высокая производительность для поиска по связям и анализа графов.

Хорошо подходят для социальных сетей, рекомендационных систем, управления сетями и анализа данных.

5. **Ключ-значение базы данных**

Базы данных типа "ключ-значение" хранят данные в виде пар "ключ-значение". Это простейший вид базы данных, оптимизированный для быстрого доступа по ключу.

**. Ключ-значение база данных: Этот тип баз данных хранит данные в виде пар ключ-значение и обеспечивает быстрый доступ к данным по ключу.**

**Примером такой базы данных является Redis.**

Примеры: Redis, DynamoDB, Memcached, Riak.

Особенности:

Высокая производительность для операций чтения и записи.

Простая модель данных.

Хорошо подходят для кэширования, управления сессиями и хранения настроек приложений.

1. **Команды**

Базовый вариант **SQL** содержит порядка **40 команд** (часто еще называемых запросами или операторами).

Ниже они представлены в порядке обязательности владения для тестировщика (от обязательных к желательным)

**Операторы DML (***Data Manipulation Language***) - операторы манипулирования и изменения данных**

**SELECT** - отобрать строки из таблиц

**INSERT** - добавить строки в таблицу

**UPDATE** - изменить строки в таблице

**DELETE - удалить строки в таблице   
  
Операторы DDL (***Data Definition Language***) -  определения объектов базы данных, с их помощью изменяется структура БД:**

**CREATE DATABASE** - создать базу данных

**DROP DATABASE** - удалить базы данных

**CREATE TABLE** - создать таблицу

**ALTER TABLE** - изменить таблицу

**DROP TABLE** - удалить таблицу

**CREATE  DOMAIN** - создать домен

**ALTER DOMAIN** - изменить домен

**Операторы DCL (***Data Control Language***) - защиты и управления данными**

**GRANT** - предоставить привилегии пользователю или приложению на манипулирование объектами

**REVOKE** - отменить привилегии пользователя или приложения

**Команды управления транзакциями TCL (***Transaction Control Language***)**

Несколько DML-команд могут быть объединены в единое логическое целое, называемое транзакцией.

**COMMIT** - завершить транзакцию и зафиксировать все изменения в БД

**ROLLBACK** - отменить транзакцию и отменить все изменения в БД

**SET** **TRANSACTION** - установить некоторые условия выполнения транзакции

**CREATE TABLE**

Этот оператор используется для создания новой таблицы в базе данных.

Он определяет структуру таблицы, включая названия столбцов, их типы данных и ограничения.

Для каждого поля (столбца) нужно обязательно указать название и тип данных.

**Основные типы данных:**

**Целочисленные типы данных:**

* **INTEGER** (**INT**) - от -2147483648 до 2147483647
* **SMALLINT** - от -32768 до 32767
* **BIGINT** -9,223,372,036,854,775,808 до 9,223,372,036,854,775,807

**Дробные числа:**

* **DECIMAL** - дробное число, хранящееся в виде строки
* **FLOAT** - число с плавающей точкой. Например, 6,5

**Строковые типы данных:**

* **VARCHAR(n)** - символьная строка n длины

**Тип данных для даты и времени:**

* **DATE** - Хранит календарную дату (год, день и месяц) без времени суток
* **TIMESTAMP** - Хранит дату и время с учетом и без учета часового пояса сервера базы данных.

**Булев тип данных:**

* **BOOLEAN** - хранит логическое булево значение (true/false)
* **INSERT INTO:**
* Этот оператор используется для добавления новых записей в таблицу.
* Для каждого поля (столбца) нужно обязательно указать значение. Строковые типы данных при этом помещаются в кавычки.

**UPDATE:**

* Этот оператор используется для обновления существующих записей в таблице.
* Синтаксис команды прост:  
  **UPDATE** имя\_таблицы **SET** название\_поля = новое\_значение\_поля **WHERE** id = номер
* **DELETE FROM:**
* Этот оператор используется для удаления записей из таблицы.

**Основные (часто используемые) команды и символы для SQL запроса:**

1. **SELECT** - выборка данных из таблицы

2. **\*** - все столбцы

3. **FROM** - указание таблицы, из которой выбираются данные

4. **WHERE** - условие для фильтрации данных

5. **AND** – логическое строгое "И"

6. **OR** - логическое "ИЛИ"

7. **NOT** - логическое "НЕ”

8. **=** - оператор равенства

9. **<>** или **!=** - оператор неравенства

10. **>** - оператор больше

11. **<** - оператор меньше

12. **>=** - оператор больше или равно

13. **<=** - оператор меньше или равно

14. **LIKE** - оператор для сравнения с шаблоном

15. **ORDER** **BY** - сортировка результатов

15.1 **GROUP BY –** группировка результатов

16. **ASC** - по возрастанию

17. **DESC** - по убыванию

18. **LIMIT** - ограничение количества возвращаемых строк

19. **()** – обособление части запроса для корректного выполнения

20. **AS** – задает полю псевдоним. Например, ’name **AS** имя’

21.**DISTINCT** используется в языке SQL для удаления дубликатов из результирующего набора данных

22. **BETWEEN** -  позволяет выбрать строки, удовлетворяющие условию, когда значение столбца находится в определенном диапазоне.

**Агрегатные функции и группировка в SQL:**

1. **COUNT()** - функция для подсчета количества строк или значений

2. **SUM()** - функция для вычисления суммы значений

3. **AVG()** - функция для вычисления среднего значения

4. **MAX()** - функция для нахождения максимального значения

5. **MIN()** - функция для нахождения минимального значения

6. **GROUP BY** – оператор группировки данных

Оператор **WHERE** не может быть использован для фильтрации агрегатных функций!

Для этого был придуман оператор **HAVING**

Оператор **HAVING** аналогичен команде **WHERE**за тем исключением, что применяется не для всего набора строк таблицы, а для набора созданного оператором **GROUP BY**и применяется всегда строго после него. То есть **HAVING** фильтрует группа, а не строки.

**JOIN**:

Этот оператор используется для соединения таблиц между собой. Как правило таблицы соединяются между собой по первичному и внешнему ключу. Но можно соединять таблицы и по столбцам, которые не являются ключами.

**Первичный ключ** – Это поле в таблице, которое не может содержать дубликатов значений. Первичный ключ используется для установления связей между таблицами и для обеспечения ссылочной целостности данных и представляет собой уникальный идентификатор каждой записи в таблице.

**Внешний ключ** - это поле в таблице, которое ссылается на первичный ключ в другой таблице. Используется для установления связей между двумя таблицами.

Синтаксис команды:

**SELECT** \* **FROM** table\_1 as t1

**JOIN** table\_2 as t2 **ON** **t1.column =  t2.column**

**LEFT JOIN** возвращает все записи из левой таблицы и совпадающие записи (если есть) из правой таблицы.

**RIGHT JOIN** возвращает все записи из правой таблицы и совпадающие записи (если есть) из левой таблицы

**INNER JOIN (JOIN)** возвращает все записи, которые имеют совпадающие значения в обеих таблицах.

**FULL OUTER JOIN (FULL JOIN)** возвращает все записи, которые имеют совпадающие значения в обеих таблицах. При этом, возвращает также данные из обеих таблиц, которые не имеют совпадающие значения.

1. **Тестирование Мобильных приложений**

Мобильные приложения — это программы, которые запускаются на мобильных устройствах, таких как смартфоны и планшеты, телевизоры, умные часы и прочее

Нативное мобильное приложение

**"Native" с английского — «родной». Нативное мобильное приложение** — это приложение, которое создается под конкретную платформу. Нативное мобильное приложение написано на «родном» для платформы языке программирования: для Android — Kotlin и Java, для Apple iOS — Objective-C и Swift.Стандартные элементы дизайна и интерфейса, характерные для конкретной платформы, создаются с помощью специальных инструментов и компонентов, предоставляемых каждой платформой.

Нативное мобильное приложение имеет доступ ко всем нативным технологиям и аппаратным возможностям конкретной платформы**. Instagram, watsapp,**

Плюсы:

Высокая производительность: приложения работают напрямую с аппаратными ресурсами, что обеспечивает плавную работу, быструю загрузку и отзывчивый интерфейс.

Полный доступ к функциям устройства: приложения могут использовать все возможности устройства, такие как камера, GPS, микрофон, хранилище и другие функции.

Лучший пользовательский опыт: приложения используют стандартные элементы дизайна и интерфейса, что делает их более привычными и интуитивно понятными для пользователей.

Улучшенная безопасность: Нативные приложения разработаны с учетом требований безопасности конкретной платформы, что делает их более защищенными от угроз.

Более качественные игры и мультимедийные приложения: приложения обеспечивают более высокую производительность и плавность работы для требовательных к ресурсам игр и мультимедийных приложений.

Минусы:

Более высокая стоимость разработки: необходимо разрабатывать отдельный код для каждой платформы, что увеличивает стоимость и время разработки.

Более сложное обновление: обновления требуют публикации в магазинах приложений, что может занимать время и требовать дополнительных процедур.

Необходимость специалистов: требуются разработчики с опытом работы с конкретными платформами (iOS или Android)

**Гибридное мобильное приложение** - это тип мобильного приложения, которое сочетает в себе элементы как нативных, так и веб-приложений. Оно разрабатывается с использованием веб-технологий (HTML, CSS, JavaScript) и упаковывается в нативный оболочку, которая позволяет ему работать на разных мобильных платформах (Android, iOS) без необходимости отдельной разработки для каждой. **Uber, google maps, ebay**

**Плюсы:**

**Более низкая стоимость разработки:**

Можно  использовать один  код  для  нескольких  платформ,  что  сокращает  стоимость  и  время  разработки.

**Более быстрое развертывание:**  Обновления  можно  выпустить  непосредственно  через  веб-сервер,  что  делает  их  более  оперативными.

**Легко  адаптировать  к  разным  платформам:**   приложения  могут  легко  изменяться  под  различные  размеры  экрана  и  особенности  платформ.

**Минусы:**

**Низкая производительность**:  Гибридные  приложения  работают  в  веб-вью,  что  может  привести  к  снижению  производительности,  особенно  при  использовании  требовательных  к  ресурсам  функций.

**Ограниченный доступ к функциям устройства**:  Доступ  к  некоторым  функциям  устройства  может  быть  ограничен  или  требовать  использования  плагинов.

**Менее привлекательный пользовательский опыт**:  Гибридные  приложения  могут  не  выглядеть  и  не  работать  так  же  плавно,  как  нативные.

**Гибридное мб – тип мобильного приложения, которое сочетает в себе элементы как нативных, так и веб-приложений.**Веб-технологии: HTML, CSS, JavaScript  +  нативная оболочка,   
Может работать на разных мобильных платформах (Android, iOS)   
Нет необходимости отдельной разработки для каждой.

**Мобильное веб-приложение -** это веб-сайт, оптимизированный для просмотра на мобильных устройствах. Оно работает в веб-браузере и не требует загрузки и установки.

**Плюсы:**

**Доступность:** Работают на всех устройствах с браузером.

**Простая разработка:** Используют стандартные веб-технологии.

**Легкое обновление:** Автоматическое обновление без необходимости скачивания.

**Низкие затраты:** Более дешевая разработка и поддержка, чем у нативных приложений.

**Быстрая загрузка:** Не требуется загрузка и установка, что делает запуск более быстрым.

**Минусы:**

**Ограниченная функциональность:** Мобильные веб-приложения не имеют доступа к функциям устройства, таким как камера, GPS и сенсоры, без использования API.

**Худшая производительность:** Могут работать медленнее, чем нативные приложения, особенно на слабых устройствах.

**Проблемы с оффлайн-доступом:** Работают только при наличии интернет-соединения.

**Сложность с персонализацией:** Менее персонализированный опыт, чем у нативных приложений.

**Ограничения по дизайну:** Дизайн может быть ограничен

**Сложность тестирования мобильных приложений** заключается в том, что нужно проверить работоспособность программы на самых разных типах устройств. Если на одном смартфоне приложение функционирует нормально, то это не означает, что на другом не возникнет ошибок. Для этого специалисту нужно работать со статистическими данными, чтобы протестировать программу на самых популярных устройствах. Необходимо проверить приложение на различных версиях мобильных ОС, учитывая:

* разнообразие девайсов: необходимо учитывать классы эквивалентности по устройствам и создавать матрицу разрешений

( челки, разрешения и размеры,складные телефоны,специфические комплектующие, влияющие на работу устройства : жесты, стилусы, доп. клавиатуры);

* операционную систему и её версию;
* характеристики процессора;
* объем оперативной и встроенной памяти;

     -       частое обновление приложений;

     -       необходимость иметь парки живых

устройств (эмуляторов недостаточно)

1. **Жизненный цикл Android**

1. Создание (onCreate):

- Вызывается, когда Activity создается впервые.

- Здесь происходит инициализация компонентов, создание интерфейса, загрузка данных и т.д.

2. Старт (onStart):

- Вызывается, когда Activity становится видимой для пользователя.

- Здесь можно начать анимацию, запустить звуки

3. Возобновление (onResume):

- Вызывается, когда Activity находится на переднем плане и может взаимодействовать с пользователем.

- Здесь можно получить фокус ввода, начать обновления данных и т.д.

4. Пауза (onPause):

- Вызывается, когда Activity частично скрыта другим Activity, но все еще видна.

- Здесь нужно освободить ресурсы, которые могут быть ресурсоемкими (например, остановить анимацию, музыку).

5. Остановка (onStop):

- Вызывается, когда Activity полностью скрыта от пользователя.

- Здесь можно сохранить данные Activity и освободить ресурсы, которые больше не нужны.

6. Восстановление (onRestart):

- Вызывается, когда Activity снова становится видимой после остановки.

- Здесь можно восстановить состояние Activity.

7. Уничтожение (onDestroy):

- Вызывается, когда Activity окончательно уничтожается.

- Здесь нужно удалить все данные, освободить ресурсы и завершить все задачи, связанные с Activity.

**Жизненный цикл приложения iOS**

Жизненный цикл iOS-приложения – это последовательность этапов, которые приложение проходит, начиная с момента запуска и заканчивая моментом его закрытия  
  
 1. Not running: Приложение не запущено  
На этом этапе приложение не запущено и не выполняется на устройстве. Этот этап может быть вызван различными причинами, такими как отсутствие установки приложения на устройстве или закрытие приложения пользователем.  
  
 2. Inactive: Приложение активно, но не работает  
Когда пользователь запускает приложение, оно переходит в состояние Inactive. На этом этапе приложение находится в фоновом режиме и не выполняет никаких действий. В это время, пользователь может сделать что-то еще, включая переключение на другое приложение.  
  
 3. Active: Приложение активно и работает  
Когда пользователь переключается на приложение, оно переходит в состояние Active. На этом этапе приложение полностью активно и работает, выполняя задачи, запущенные пользователем.  
  
 4. Background: Приложение работает в фоновом режиме  
Когда пользователь переключается на другое приложение, приложение переходит в состояние Background. На этом этапе приложение продолжает выполнять задачи в фоновом режиме, такие как загрузка данных или проигрывание музыки.  
  
5.  Suspended: Приложение находится в фоновом режиме, но не выполняет действий  
Если приложение не было запущено в течение некоторого времени, оно переходит в состояние Suspended. На этом этапе приложение находится в фоновом режиме, но не выполняет никаких действий и не занимает системные ресурсы.  
Terminated: Приложение было закрыто

**Основные проверки**

жизненный цикл (процесс установки/удаления/обновления/сворачивания/приема звонка/выключения и тд)

- разные ПО и версии

- анализ процесса установки/удаления

- тестирование возможности обновлений

- пермишены (разрешения доступа)

- пуш нотификации

- лок кнопок

- работоспособность в разных режимах (онлайн, оффлайн,ВПН)

- эррорс хендлинг (нотификации об ошибках интернета,сервера

- юзабилити (кнопки, настройки,навигация)

- гайдлайны https://droider.ru/post/filosofiya-ios-vs-android-razbor-17-07-2022/

- сетевой трафик (реакции МП на ошибки сервера/сети,отправке запросов и получении ответов)

- UI (отображние элементов на разных разрешениях,могут поехать, могут отсутствовать/не помещаться)

- тестирование функций приложения (авторизации, гео, листинга,чекаута и тд)

- тестирование специфический для устройства функций

- интернет (прерывание, таймаут/задержки, разные скорости)

- жесты https://stfalcon.com/ru/blog/post/intuitive-gestures-in-mobile-app-design

- диплинки

- аксессобилити

- перфоманс (несколько запросов на один запрос нагружает систему)

**Инструменты**

Снифферы (Charles Proxy, Fiddler)

- Симуляторы виртуально воспроизводят поведение системы и её интерфейса. Симуляции имитируют выполнение кода. В большинстве случаев, для запуска симулятора, используются XCode или Android Studio.

- Эмуляторы воссоздают все основные компоненты устройства, в том числе процессор, память и устройства ввода/вывода

- DevTools для тестирования адаптивных версий приложения

- Тестфлайт

-BrowserStack - это облачная платформа для тестирования веб-приложений и мобильных приложений. Она предоставляет доступ к широкому спектру браузеров, операционных систем и устройств, что позволяет разработчикам и тестировщикам проверять свои приложения в различных средах