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. **Как тестировать требования, какие критерии?**

**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. **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. **Принципы тестирования**

* **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. **С какими видами тестирования ты работаешь?**

**1.Функциональные виды тестирования: базируются на функциях и особенностях, а также взаимодействии с другими системами  
- интеграционное Проверяем взаимодействие между компонентами внутри системы; да  
- системное выполняемое на полной, интегрированной системе;**

**- модульное**

**2.Нефункциональные виды тестирования:   
- тестирование стабильности или надежности; нет  
- тестирование удобства пользования UX UI | кроссбраузерность. Да**

**3.Виды тестирования связанные с изменениями: да  
- дымовое;  
- регрессионное;  
- санитарное.**

**По знанию системы :**

**-белого ящика**

**-серого ящика**

**-черного ящика**

**По позитивности :**

**- позитивные**

**-негативные**

**По исполнителям :**

- альфа (отдельная команда тестеров)

-бета тесты

**По степени автоматизации :**

**- ручное**

-автоматическое

1. **Критерии успешного релиза?**

**Функциональность: Продукт должен предоставлять все основные функции и возможности, которые были запланированы и обещаны. Все функции должны работать должным образом, без критических ошибок и непредвиденного поведения.**

**Качество: Продукт должен быть стабильным, надежным и исправным. Ошибки и сбои должны быть минимальными, и если они возникают, они должны быть быстро исправлены с помощью обновлений или патчей.**

**Производительность: Продукт должен работать эффективно и обеспечивать достаточную производительность для удовлетворения потребностей пользователей. Он не должен быть излишне медленным или затратным по ресурсам.**

**Интерфейс и удобство использования: Пользовательский интерфейс должен быть интуитивно понятным и привлекательным для целевой аудитории. Продукт должен быть легко освоим и удобным в использовании без лишних сложностей или запутанности.**

**Безопасность: Продукт должен быть защищенным от угроз и обладать должным уровнем безопасности. Конфиденциальность данных пользователей должна быть обеспечена, а система должна быть устойчивой к взломам и злоумышленникам.**

**Поддержка и обслуживание: Команда разработчиков должна предоставлять надлежащую поддержку пользователей после выпуска продукта. Они должны быть доступны для решения проблем, ответов на вопросы и предоставления обновлений и исправлений.**

**Удовлетворенность клиентов: В конечном итоге, успешный релиз определяется удовлетворенностью и положительными отзывами пользователей. Продукт должен решать их проблемы и соответствовать их ожиданиям.**

**все задачи соответствуют требованиям,**

**нет багов выше уровня минор,**

**регресс проведен**

1. **Какие бывают требования?**

**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

**Функциональные требования.**  Описывают функциональные возможности продукта, а также поведение системы в различных ситуациях**.**

**Нефункциональные требования.** Описывают качественные характеристики продукта: удобство использования, безопасность, надёжность, производительность, масштабируемость и другие атрибуты.

1. **КАК БЫЛ УСТРОЕН РЕЛИЗ ? Разница релиза к в монолитный сайт и в микросервисный?**

1. Планирование релиза  
- Определение целей релиза: какие функции, исправления и улучшения будут включены в новую версию.  
- Создание дорожной карты: планирование сроков и этапов разработки, тестирования и развертывания.  
  
2. Разработка  
- Команда разработки работает над реализацией новых функций и исправлением ошибок, согласно требованиям и плану, установленным на этапе планирования.  
- Использование систем контроля версий (например, Git) для управления изменениями в коде.  
  
3. Тестирование  
- Тестирование на этапе разработки: Разработчики могут выполнять юнит-тесты и интеграционные тесты, чтобы убедиться, что изменения работают корректно.  
- Регрессионное тестирование: Проверка, что новые изменения не нарушили существующий функционал.  
- Приемочное тестирование: Оценка новой версии со стороны конечных пользователей или QA-команды, чтобы удостовериться, что она соответствует требованиям.  
  
4. Подготовка к релизу  
- Создание релизной документации: (план, сценарии, отчет , требования к системе) описание новых функций, изменений, исправлений и известных проблем.  
- Уведомление всех заинтересованных сторон о предстоящем релизе и его содержании.  
  
5. Развертывание  
- Процесс развертывания новой версии на продуктивной среде. Это может включать:  
- Деплой: перенос новых файлов, конфигураций и данных на сервер.  
- Миграция данных: выполнение необходимых изменений в базе данных.  
- Проверка работоспособности: выполнение тестов для подтверждения корректности работы после развертывания.  
  
6. Мониторинг и поддержка  
- После релиза команда отслеживает работу приложения, чтобы выявить возможные ошибки или проблемы.  
- Обеспечение поддержки пользователей и реагирование на их отзывы.  
  
7. Ретроспектива  
- После завершения релиза команда проводит анализ, чтобы обсудить, что прошло хорошо, а что можно улучшить в будущем. Это помогает оптимизировать процесс для следующих релизов.

**Как был устроен релиз?** У нас был двух недельный **спринт**, Первый день: получение задач, изучение задач, вычитка документаций, планирование, делили задачи с коллегой. Далее брал задачи из **бэклога** (список задач), и тестировал, каждый день у нас были **дейлики** по 15 мин - обсуждали что делали вчера, что буду делать сегодня, какие есть сложности. По четвергам на 2 неделе был **грумминг** - Планирование будущих задач, знакомство с документацией, проектирование, планирование. **Ретроспективы** по пятницам на 2 неделе - анализировали что получилось, что не получилось, делились ОС, цели на следующий спринт. На второй неделе в идеале в среду начинали делать **регресс,** а далее если не было багов выше минор и овнер согласовывал запуск то катили в прод - Задачи на регресс и на прод готовил Тимлид.

Если задают вопрос **кто катил задачу на тестовый стенд** - я брал задачу, нажимал start testing, прокидывалась ссылка в задачу (у нас была настроена интеграция jira с гитлабом) - переходил по ссылке, меня перекидывало в гитлаб ci/cd - в мой пайплайн, ждад когда закатится джоба(сценарий автоматизированной сборки) на тестовый стенд (stage), если все - ок: шел тестировать, если нот ок: привлекал разработчика, если разработчик не понимал в чем проблема, тогда я шёл к дувопсу, он всегда мне помогал)

Если задают вопрос: **нашла баги - сразу заводила? Нет,** сначала шла к разработчику и показывала проблему, если её удавалось решить быстро (до 30 минут), то сразу правили, если же понимали что большая проблема, то заводила баг.

**Микросервисы можно параллельно обновлять и заливать функциональность, в монолит последовательно**

**ЛОГИ**

1. **Что такое логи?**

**Логи** — это файлы или записи, содержащие информацию о событиях, операциях или действиях, которые происходят в системе, приложении или сети. Логи создаются автоматически различными компонентами системы, серверов, приложений или устройств и являются важным инструментом для мониторинга, отладки и анализа работы системы.

**Логи приложений (Application logs).** Это логи, создаваемые самим приложением в процессе его работы. Это может быть веб, десктоп и мобильное приложение. Они содержат информацию о выполнении операций, событиях, ошибочных ситуациях, запросах, ответах и других событиях, внутри приложения.

**Логи сервера (Server logs**). Это логи, генерируемые серверами и веб-серверами. Они содержат информацию о работе сервера, запросах, ошибочных ситуациях, подключениях и других событиях, происходящих на сервере.

**Системные логи (System logs**): Это логи, записываемые операционной системой. Они содержат информацию о работе операционной системы, событиях, ошибках, состоянии системы, процессах, сетевых подключениях и других системных событиях.

1. **Зачем нужны?**

 **Диагностика и отладка**: Логи помогают разработчикам и администраторам находить и исправлять ошибки, отслеживать поведение системы и приложения.

 **Аудит**: Логи служат источником информации о действиях пользователей и системе для анализа активности и расследования инцидентов.

 **Мониторинг производительности**: Логи позволяют отслеживать производительность системы или приложения в реальном времени, выявляя узкие места или сбои.

 **Безопасность**: Анализ логов помогает обнаруживать подозрительные действия, взломы, DDoS-атаки и другие угрозы.

1. **Уровни логирования?**

**Пример иерархии уровней логирования:**

* **FATAL — самые важные и критические ошибки.**
* **ERROR — ошибки, но не критичные для работы приложения.**
* **WARN — предупреждения о возможных проблемах.**
* **INFO — информационные сообщения о нормальной работе системы.**
* **DEBUG — детальная информация для отладки.**
* **TRACE — самый подробный уровень, включающий каждую мелочь в работе программы.**

**Когда тестировщик смотрит в логи**

**При тестировании новой фичи.** Стоит держать логи открытыми, они помогут отследить правильность выполнения определенных операций, последовательность событий и другие аспекты функциональности приложения.

**К примеру, если упадет ошибка (ERROR) или предупреждение (WARN)**— мы всегда могли сообщить разработчику и быстро нейтрализовать проблему.

**При релизе приложения и проведении регрессионного тестирования** нужно проверять логи, часто в командах это делают **разработчики.**

**При проверке взаимодействия с внешними системами, такими как базы данных, API или другие службы.** Логи могут содержать информацию о запросах и ответах, передаваемых данным, кодах состояния и других деталях взаимодействия.

**Для более точного определения причины бага и отладки ошибки.** В логах может содержаться необходимая информация для идентификации бага, логи могут содержать полезную информацию о возникших исключениях, трассировке стека, сообщениях об ошибках или недостаточных данных, которые могут помочь воспроизвести и понять причину ошибки. А чем больше информации мы принесём разработчику, тем быстрее он починит баг.

**Для оценки производительности приложения.** Логи могут содержать информацию о времени выполнения определенных операций, использовании ресурсов (например, процессора, памяти) и других показателях производительности. Тестировщик может использовать эти данные для определения узких мест в приложении и предложения улучшений.

**При работе с событиями системы.** Вас могут попросить разработчики или вам самим потребуется более детальная информация для понимания работы системы.

1. **Какие уровни логирования нужны тестировщику?**

**Эррор или дебаг - например, обратился пользователь, что оплатил, а оплата не прошла у него, я проверяю, что он действительно оплатил, но надо понять, что произошло и прошу разраба включить делоги на оплату заказа, а так обычно не засоряют дебагами**

1. **Что проверял в логах?**

**клиент указал некорректный номер при оплате, будет ли уходить смс на несуществующий номер**

1. **Где располагались файлы с логами на твоем проекте?**

**в кибане через разработчиков работали, а в даталоге были настроены фильтры**

1. **Что смотрел в кибане, как с ней работал?** **не смотрел сам, через разработчиков**
2. **Что смотрел в Датадоге, как с ним работал? а в даталоге были настроены фильтры**
3. **Что такое трейсы в логах?**

**Трейс или трейсбек (от англ. traceback) - сообщение об ошибке в программе, по которому можно отследить, в каком именно месте кода возникла ошибка. Если у тебя игра внезапно закрывается и открывается блокно с файлом traceback.**

1. **Приведи последний пример что делал в логах**

**проверял, как логируется событие записи данных в базу данных, по клиенту и по заказу,**

1. **Отличия тестирования мобилок от веба?**

Тестирование мобильных приложений и веб-приложений отличается по ряду параметров, связанных с особенностями платформ, способов взаимодействия пользователей и техническими ограничениями.

1. **Операционные системы и устройства**

Мобильные приложения: Тестируются на различных операционных системах (iOS, Android) и их версиях. Мобильные устройства имеют разные экраны, производительность, возможности сети и физические характеристики (размер экрана, наличие аппаратных кнопок).

Веб-приложения: Обычно тестируются на различных веб-браузерах (Chrome, Firefox, Safari и др.) и их версиях, а также на разных операционных системах (Windows, macOS, Linux).

**2. Разновидности приложений**

* **Мобильные приложения** бывают трех типов:
  + **Нативные**: разработанные для конкретной операционной системы (iOS, Android).
  + **Кроссплатформенные**: работающие на нескольких ОС с единой кодовой базой (React Native, Flutter).
  + **Гибридные**: это веб-приложения, встроенные в мобильное приложение (например, с использованием WebView).
* **Веб-приложения** могут быть как обычными веб-сайтами, так и сложными SPA (Single Page Applications), где происходит активное взаимодействие на стороне клиента.

**3. Интерфейс и взаимодействие**

* **Мобильные приложения**:
  + Основное взаимодействие идет через сенсорный экран (тачскрин), что требует проверки жестов (тапы, свайпы, мультитач).
  + Возможны прерывания во время работы приложения (звонки, уведомления, низкий заряд батареи), что также нужно тестировать.
  + Разные ориентации экрана (портретная и ландшафтная) требуют адаптации интерфейса.
  + Поддержка работы оффлайн — некоторые приложения должны сохранять свою функциональность без подключения к интернету.
* **Веб-приложения**:
  + Взаимодействие в основном через клавиатуру и мышь.
  + Меньше зависимостей от аппаратных функций, таких как камера, GPS, акселерометр и т.д., но важно протестировать респонсивность (адаптивность интерфейса) на разных размерах экранов (особенно на мобильных браузерах).
  + Перерывы в работе менее критичны, так как веб-приложения обычно работают только при наличии подключения к сети.

**4. Средства тестирования**

* **Мобильные приложения**:
  + Тестирование на реальных устройствах и эмуляторах/симуляторах.
  + Необходимость тестирования на широком спектре устройств с разными характеристиками (разрешение экрана, версия ОС, производительность).
  + Используются специфические инструменты, такие как Appium, Espresso (для Android), XCUITest (для iOS), а также облачные платформы для тестирования на реальных устройствах (например, BrowserStack, Sauce Labs).
* **Веб-приложения**:
  + В основном используются инструменты для тестирования веб-браузеров, такие как Selenium, Cypress.
  + Тестирование можно проводить с помощью браузерных расширений и инструментов разработчика.
  + Легче эмулировать разные браузеры и устройства.

**5. Производительность**

* **Мобильные приложения**:
  + Тестируются на производительность при ограниченных ресурсах устройства (батарея, оперативная память, процессор).
  + Важно тестировать поведение приложения при слабом или прерывистом интернет-соединении, влиянии фоновых процессов на работу приложения.
  + Проверка потребления трафика, особенно для пользователей с лимитированным интернетом.
* **Веб-приложения**:
  + Производительность зависит от браузера, однако тесты направлены на загрузку страниц, рендеринг, скорость работы скриптов.
  + Меньше проблем с управлением ресурсами, но нужно тестировать скорость работы на разных типах интернет-соединений.

**6. Обновления и деплой**

* **Мобильные приложения**:
  + Обновления проходят через магазины приложений (App Store, Google Play), что добавляет задержки и сложности при распространении новой версии.
  + Нужно учитывать поддержку старых версий приложения, так как пользователи не всегда сразу обновляют свои приложения.
* **Веб-приложения**:
  + Обновления происходят практически мгновенно, поскольку код и ресурсы хранятся на сервере. Пользователи видят изменения сразу при обновлении страницы.

1. **Как проходил процесс тестирования мобилок?**
2. **Работал ли с андроид студио и что делал? да**
3. **В каких случаях делали подмену значений? Что проверяли?**

Подмена значений что-бы убрать рассрочку в оплате.

1. **Что знаешь про активити Андройда?**

**1. Создание (Creation): В этом этапе активити создается и инициализируется. Вызывается метод onCreate(), где происходит установка контента активити, инициализация переменных и настройка интерфейса.   
  
2. Запуск (Start): После создания активити она переходит в состояние "запущена". Вызывается метод onStart(), где происходит подготовка к отображению активити на экране.   
  
3. Возобновление (Resume): Активити становится видимой для пользователя и получает фокус. Вызывается метод onResume(), где происходит запуск анимаций, запуск воспроизведения мультимедиа и другие операции, связанные с отображением активити.   
  
4. Приостановка (Pause): Активити теряет фокус, но останется видимой для пользователя. Вызывается метод onPause(), где происходит сохранение состояния активити и временное приостановление выполнения операций.   
  
5. Остановка (Stop): Активити становится невидимой для пользователя. Вызывается метод onStop(), где происходит освобождение ресурсов и сохранение данных.   
  
6. Уничтожение (Destroy): Активити полностью уничтожается и освобождает все ресурсы. Вызывается метод onDestroy(), где происходит освобождение памяти, закрытие соединений и другие завершающие операции.   
  
Важно отметить, что активити может переходить между этапами в зависимости от действий пользователя или системных событий. Например, если пользователь переключается на другое приложение, активити может быть приостановлена или остановлена. При возвращении к приложению активити может быть возобновлена.**