Финальные вопросы QA

Вводные

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. **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 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. **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. **Что делает аналитик на проекте?**

**Роль аналитика на проекте разработки программного обеспечения (ПО) включает в себя ряд задач, связанных с анализом требований, пониманием потребностей заказчика и обеспечением связи между заказчиком и командой разработки.**

1. **Что делает проджект менеджер? Проект-менеджер играет ключевую роль в управлении проектами от начала до конца. Он отвечает за планирование, координацию и контроль всех аспектов проекта с целью достижения поставленных целей в заданных рамках времени и бюджета.**
2. **Кто такой продукт овнер?**

**Продукт-овнер (Product Owner) — это ключевая роль в методологии управления проектами Scrum, применяемой в разработке программного обеспечения. Основная задача продукт-овнера заключается в максимизации ценности продукта, которую команда разработчиков создает.**

1. **Что такое манки тестинг или ад хок тестирование?**

### **Манки тестинг** — это более хаотичное и случайное тестирование, направленное на выявление ошибок при непредсказуемых действиях пользователя.

### **Ад-хок тестирование** — это свободное, но целенаправленное тестирование, основанное на знаниях и опыте тестировщика.

### Оба метода полезны как дополнение к формальным методам тестирования, помогая находить ошибки, которые могут быть упущены при обычном тестировании по заранее подготовленным сценариям.

1. **Какие есть тестовые стенды и как называются? Почему их несколько?**

**Тестовые стенды** — это изолированные среды, используемые для разработки, тестирования и проверки программного обеспечения перед его выпуском в эксплуатацию. Обычно их несколько, потому что каждая среда решает определенные задачи на разных этапах разработки и тестирования. Каждая из этих сред может отличаться по конфигурации, данным и набору проверяемых условий.

* 1. **Development (Dev): Среда разработки. Здесь разработчики внедряют новые функции, проверяют изменения в коде и проводят базовое тестирование.**
  2. **QA/Test: Среда для ручного или автоматизированного тестирования (Quality Assurance). Здесь тестировщики проверяют функциональные и нефункциональные аспекты приложения.**
  3. **Production: Боевая среда, которая доступна пользователям. В production выпускается стабильная версия системы после успешного прохождения всех тестов.**

1. **В каком случае ты взаимодействуешь с аналитиком?**

**"QA-инженер (специалист по обеспечению качества) и аналитик могут взаимодействовать на проекте по разработке программного обеспечения в различных случаях, например:  
  
Понимание требовани  
  
Проверка документации  
  
Уточнение критериев приемки  
  
Разрешение непониманий или несоответствий  
  
Обратная связь по результатам тестирования  
  
В целом, взаимодействие между QA-инженером и аналитиком помогает обеспечить более полное понимание требований и критериев качества продукта, что в свою очередь способствует успешной разработке и выпуску программного обеспечения."**

1. **Ты QA инженер или тестировщик?**

**Тестировщик  
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. **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. **Из каких этапов состоит ЖЦ ПО**

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

1. **Какие две стратегии разработки выделяют ?**

**Существует множество стратегий разработки программного обеспечения, но две из самых распространенных и широко используемых это:  
  
Каскадная стратегия разработки (Waterfall): Это классическая стратегия, которая предполагает линейное выполнение этапов разработки, начиная с определения требований и заканчивая тестированием и внедрением. Каждый этап завершается до начала следующего, и изменения в требованиях часто затруднены. Эта модель подходит для проектов с четко определенными требованиями и стабильной средой.  
  
Итеративная стратегия разработки (Iterative): Эта стратегия предполагает разбиение процесса разработки на итерации, каждая из которых включает в себя циклы разработки, тестирования и обратной связи. Каждая итерация добавляет новый функционал или улучшает существующий, и изменения могут быть легко внесены на любом этапе. Эта модель подходит для проектов с неопределенными или изменяющимися требованиями, так как она позволяет быстро реагировать на изменения и обратную связь.**

### Когда использовать спиральную модель?

1. Когда требуется частое изменения.
2. Когда проект большой.
3. Когда требования нечеткие и сложные.
4. Когда изменения могут потребоваться в любое время.
5. Крупные и высокобюджетные проекты.
6. **Что знаешь про гибкий подход к разработке?**

**AGILE — ЭТО ФИЛОСОФИЯ, СЕМЕЙСТВО ГИБКИХ ПОДХОДОВ К РАЗРАБОТКЕ ПО**

**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. **Что делать, если нет тестовой документации на проекте?**

**Обратиться к аналитику и начать вести**

1. **Что делать, если задача пришла без требований?**

**Посмотреть реализацию подобных выполненных ранее задач, поискать в аналтике, обратиться к аналитику, разработчику, дизайнерам**

1. **В каком случае ты взаимодействуешь с дизайнером?**

**Вопросы, касающиеся макетов сайта, если есть замечания**

1. **Как ты тестировал UX дизайн?**

**UX -Тест на удобство использования также называется тестом на удобство использования, который используется для оценки удобства использования приложения . Идея внедрения тестирования удобства использования заключается в оценке простоты и гибкости использования приложения с точки зрения пользователя.**

**На это не было отдельных задач, удобство тестируем по ходу дела**

1. **В какой момент ты тестируешь требования?**

**Когда задача переходит к нам в спринте или на этапе формирования требований**

1. **На каком языке у тебя на проекте был написан бэкенд?**

**Wappalyzer на нем можно пробить на чем написан бэк**

1. **От кого тебе приходили задачи на проекте?**

**От тимлида группы тестирования или ПМ**

1. **Как был устроен скрам на твоем проекте?**

**2 нед спринты, дейлики, планирование спринта в чт, ретроспектива - пт, грумминг - чт ()**

1. **Как выглядит процесс разработки ПО?**

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

1. **В чем разница ватерфола от аджайла?**

**"(Waterfall): Это классическая стратегия, которая предполагает линейное выполнение этапов разработки, начиная с определения требований и заканчивая тестированием и внедрением. Каждый этап завершается до начала следующего, и изменения в требованиях часто затруднены. Эта модель подходит для проектов с четко определенными требованиями и стабильной средой.**

1. **Методологии разработки**

**«Waterfall Model» (каскадная модель или «водопад»)  
.«V-Model»  
 «Incremental Model» (инкрементная модель)  
 «RAD Model» (rapid application development model или быстрая разработка приложений)  
 «Agile Model» (гибкая методология разработки)  
 «Iterative Model» (итеративная или итерационная модель)  
 «Spiral Model» (спиральная модель)**

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. **Кто пишет тест план на задачи спринта? Где он лежит? На какие вопросы он должен отвечать?**

**Тест-план — это документ, который детализирует конкретные действия по тестированию программного обеспечения для проекта.**

1. **Что знаешь про тестовую стратегию? Кто её определяет?**

**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.**

1. **Что знаешь про матрицу прослежиаемости требований?**

**Матрица прослеживаемости требований (RTM – Requirements Traceability Matrix) – это документ, который отображает и отслеживает требования пользователя с помощью тестовых примеров. Он фиксирует все требования, предложенные заказчиком, и контроль за их реализацией в одном документе.**

**Техники тест дизайна**

1. **Какие техники тест дизайна знаешь? Как работает каждая?**

**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. **Какие техники считаешь самыми основными?**

**Эквивалентное разделение  
Анализ граничных значений;**

1. **Поле имя может быть длиной от 2 до 20 символов, какие значения будешь проверять?**

**Позитивные: 2, 20  
Негативные - 1, 21, пустое поле**

1. **Лимит в телеграм чате 40 сообщений в час, что будешь проверять?**

**Ввести 40 сообщений сразу, проверить, что 41 не отправляется и на 61 минуте проверить, что можно снова ввести 40 сообщений**

1. **Есть кнопка загрузки файлов, допустимый только PDF до 10мб, что будешь проверять?**

**Файл 9 мб, Файл 10 мб, файл 11 мб,**

1. **Добавить в избранное товар можно только авторизованному пользователю, что проверишь?**

**Добавить с авторизацией и добавить новому пользователю.**

1. **Где применял предугадывание ошибки?**

**На различных этапах жизненного цикла разработки ПО:  
  
Анализ требований: Можно предположить, какие ошибки могут возникнуть, исходя из недостаточно ясных или противоречивых требований.   
  
Проектирование тестов: При разработке тест-кейсов тестировщик может учитывать типичные ошибки, которые могут возникнуть в различных частях приложения или на различных уровнях абстракции. .**

1. **Что делать, когда на собеседовании дали задание на тестирование поля телефон?**

**Проверить ввод цифр. Анализ граничных значений, Пробел, буквы , символы**

1. **Оставить инпут пустым, это про что?**

**Это значит оставить какое-лиюо поле для заполнения пустым**

**Чек лист и тест кейс**

1. **Что такое тест кейс?**

**Тест-кейс -(Документ, содержащий последовательность действий направленных на проверку какого-либо функционала, описывающих как прийти к ожидаемому результат )**

1. **В какой момент пишут тест кейсы?**

**В момент передачи требований или задачи**

1. **Где ведется тестовая документация?**

**Тест Айти, если мы говорим про тест кейсы и чек листы, остальное в конфлюенсе**

1. **Почему тест кейс иногда единственное место, где описаны требования?**

**Если на проекте есть проблемы с документацией, то тестеры уточняют ожидания в м2оменте и вносят их в тест кейсы.  
  
В таком случае тест-кейсы становятся основным источником информации о том, как должна работать система.**

1. **Какие правила необходимо соблюдать при написании тест кейса?**

**Каждая проверка проверяет одну функцию**

**Пишем ОР на каждый шаг**

**Опускаем ОР в тривиальных шагах**

**Пишем только итоговый ОР**

**Выбор зависит от проверки требований и других особенностей**

1. **Какие составляющие тест кейса?**

**Основные поля:  
Id (уникальный номер)  
Название  
Шаги  
Приоритет  
Ожидаемый результат   
Статус  
  
Опциональные поля:  
Тестировщик  
Ссылка на требования  
Предусловия  
Постусловия  
Окружение  
Номер сборки**

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.

1. **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.

1. **В чем ключевая разница тест кейса и чек листа?**

**ТК - это одна проверка с последовательностью шагов.  
ЧЛ - это список разных проверок**

1. **Можно ли полностью отказаться от тест кейсов?**

**не желательно, потому что есть сложных функционал, который лучше покрывать тест кейсами, где важна последовательность выполнения шагов**

1. **С каких проверок нужно начинать, валидные или невалидные?**

**Всегда с позитивных, чтобы проверить работает ли впринципе нужный функционал с валидными значениями**

1. **Что такое тест сьют?**

**Тестовый комплект, который относится к какой то конкретной функциональности, например тест сьют по корзине, каталогу, оформлению заказа. И в нем уже лежат тест кейсы**

1. **Какие пункты должны быть в чек листе?**

**Описание задачи: Четкое и краткое описание того, что должно быть сделано или проверено.   
  
Список шагов: Перечисление всех шагов или этапов выполнения задачи.   
  
Критерии успешного выполнения: Определение условий, которые должны быть выполнены для того, чтобы задача была выполнена успешно.**

1. **Как устоен процесс регресса? Как происходит запуск в кейс IO?**

**тимлид запускает, прверяем свои задачи**

1. **В регресс наборе должны быть только тест кейсы или чек листы тоже могут быть? Все**
2. **Сколько тестов в регресс наборе? 400**

**400, постоянно растут**

1. **Как и когда проводится регресс? Сколько времени занимает?**

**дня 3-4, если не один**

* 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. **Откуда берём ожидаемый результат?**

**Из требований, если их нет формируем вместе с аналитиком или заказчиком**

1. **Кто формирует требования?**

**Аналитик**

**Баг репорты и джира**

1. **Чем отличается баг от фичи?**

**Баг** — это ошибка или неисправность в программе, которая приводит к некорректному поведению системы. Баг возникает тогда, когда программа работает не так, как предполагалось изначально. Это может быть связано с логическими ошибками, неправильной реализацией требований или сбоем при взаимодействии с другими компонентами. Баги обычно требуют исправления.

Пример: приложение падает при попытке загрузить файл.

**Фича** (от англ. feature — функция) — это функциональная возможность программы, которая была запланирована и намеренно реализована разработчиками. Фичи представляют собой функционал, который пользователи могут использовать для решения своих задач. Это может быть новый интерфейс, дополнительная кнопка, фильтрация данных и т. д.

Пример: добавление функции сортировки данных в таблице по дате.

1. **Какая бывает документация на проекте?**

**Внутр документация QASE IO, TEST IT  
Чек-лист 70%  
Тест-кейс 30%  
Тест-комплект   
Тестовая стратегия   
Тест-план   
  
Внешн документация   
Баг-репорт JIRA  
Запрос на изменение   
Отчет о тестировании**

1. **Из каких элементов состоит хороший баг репорт?**

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

1. **Как писать заголовок бага? Какие блоки входят в хороший баг репорт?3**

**Заголовок Что? Где? Когда?**

1. **В каком случае нужен скринкаст?**

**Если ошибку сложно заскриншотить. Шаги воспроизведения и полученный результат лучше дополнить скринкастом.   
Если ошибка связана с графикой.**

1. **Какая бывает серьёзность? Какие есть приоритеты?**

**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) 20%-** ошибка, которая может привести к серьезным проблемам или потенциальным угрозам безопасности. Пример: При переводе денег пользователю, деньги списываются со всех счетов сразу.
* **Значительная (Major) 30% -** ошибка, которая может вызвать некритические проблемы или незначительные нарушения функциональности. Пример: При открытии истории переводов, отсутствует часть данных.
* **Незначительная (Minor) 50%-** 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. **Запрос на изменение это про что?**

**Любой участник может сформировать запрос на изменение процесса, данных, требований, продукта, оформляется таской**

1. **Какие типы задач были в джире?**

**Задача (Task): Основной тип задачи, который обычно используется для выполнения конкретных действий или заданий в рамках проекта.  
  
История (Story): Задача, описывающая новую функциональность или требование, которое должно быть реализовано в продукте.  
  
Ошибка (Bug): Задача, связанная с исправлением ошибок или дефектов в программном обеспечении.  
  
Эпик (Epic): Большой фрагмент работы, который может быть разбит на более мелкие задачи (например, истории или задачи).  
  
Подзадача (Sub-task): Мелкая задача, которая может быть частью более крупной задачи и содержать дополнительные шаги или подзадачи.**

1. **Что такое связи запроса в джире и зачем используются?**

**Связи запроса в Jira - это функциональность, которая позволяет связывать задачи и запросы между собой, чтобы показать их взаимосвязь и зависимость. Это может быть полезно в случаях, когда одна задача зависит от выполнения другой, или когда необходимо отслеживать связанные проблемы, которые имеют общую цель или требуют согласованного выполнения.**

1. **Как отслеживать исправление заведенных багов? Кто это должен делать?**

**Тестировщик должен отслеживать баг в рамках спринта и следить за его исправлением, а если этот баг не в рамках текущего спринта, то он улетает в бэклог и за него отвечает проджект-менеджер**

1. **Какая документация хранится в конфлюенсе?**

**Должны храниться все инструкции проекта, описания процессов**

1. **Кто выставляет приоритет бага?**

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

1. **Приведи пример высокого приоритета и низкой серьезности**

**Номер телефона компании на странице сайта указан неверно**

1. **Приведи пример низкого приоритета и высокой серьезности**

**Годовой финансовый отчет не работает корректно в начале года**

1. **Как формируется список тест кейсов на регресс?**

**Список тест-кейсов на регрессию формируется на основе изменений в программном обеспечении. Регрессионное тестирование направлено на проверку того, что новые изменения в коде не привели к появлению ошибок в уже существующей функциональности.   
  
Список тест-кейсов на регрессию формируется на основе изменений в коде, чтобы обеспечить полное покрытие функциональности и минимизировать риск появления ошибок после внесения изменений.**

1. **Когда находишь баги в момент проверки фичи, что делаешь?**

**Когда тестировщик обнаруживает баги в процессе проверки новой функции, важно сообщить об этом как можно скорее и принять необходимые меры:  
  
Зарегистрировать баг  
Приоритизировать баг  
Сообщить разработчикам  
Проведите дополнительное тестирование  
Отслеживайте исправление  
Документация**

1. **Где пишут тест репорт? Что в него входит, как он может выглядит?**

**Когда простые задачи, то просто в чате, либо в комментарии к задаче, либо все ок, либо вот ссылка на баг-репорт, Если задача большая, то могут создать отдельную страничку в конфлюенсе и там расписать, что найдено**

1. **Какие баги можно выпустить в прод? согласованные с заказчиком**

**API SW PM**

1. **Где были описаны API? SWAGGER**
2. **Как работали ли с коллекциями?**

**Разраб методом круд предеовал коллекцию файликом и я тестил в постмане**

1. **Как тестировали 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)  
Проверял, что API правильно логирует ошибки, связанные с некорректными типами данных. С помощью Логирования отслеживал и анализировал проблемы в производственной среде.  
  
Последнюю тестировал - получение списка рекомендованных товаров по предпочтениям**

1. **Какие статус коды чаще всего видел?**

**200 OK. Запрос успешно выполнен, в ответе содержится запрашиваемая информация.   
400 Bad Request. Запрос содержит некорректные данные или не может быть обработан сервером.   
404 Not Found. Запрашиваемый ресурс не найден на сервере.**

**500 ошибка сервера**

1. **Какие есть коды ответа API?**

**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. **Приведи примеры с какими кодами ответов API сталкивался? (3 из 200 и 3 из 400 )**

**200 OK. Запрос успешно выполнен, в ответе содержится запрашиваемая информация.   
201 Created. Запрос успешно выполнен, и в результате был создан новый ресурс.   
204 No Content. Запрос успешно выполнен, но в ответе нет содержимого.   
400 Bad Request. Запрос содержит некорректные данные или не может быть обработан сервером.   
401 Unauthorized. Требуется аутентификация для доступа к ресурсу.   
403 Forbidden. У клиента нет разрешения на доступ к ресурсу.   
404 Not Found. Запрашиваемый ресурс не найден на сервере.**

1. **Статус коды по группам + 401 403 404 201**

**401 (Unauthorized). Клиент пытается работать с закрытым ресурсом без предоставления данных авторизации. Возможно, он предоставил неправильные учётные данные или вообще ничего.   
403 (Forbidden). Сервер отказывается выполнять запрос клиента, то есть клиент не имеет необходимых разрешений для доступа.   
404 (Not Found). Сервер понял запрос, но не нашёл соответствующего ресурса по указанному URL.   
Другие статусные коды, не связанные с указанными группами:  
  
201. Подтверждает, что запрос был успешным и, как следствие, был создан новый ресурс.**

1. **Почему нужно тестировать API?**

**Тестирование API сводится к проверке бизнес-логики приложения, а также интеграционному тестированию, при выполнении которого можно выявить нестыковки в реализованной логике.**

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.

1. **Прикрутить в админку новую ручку, это про что?**

**Ручка - это просто метод публичного интерфейса приложения, будь то веб-приложение или еще какое-то. Пошло от английского handle.  
  
Пример:  
Ручки - это, например, URL'ы в API какого-нибудь сервиса.   
Вот, например, API Яндекс.Диска.   
Я могу сказать: "Чтобы загрузить файл на сервис, мне надо дёрнуть ручку https://cloud-api.yandex.net/v1/disk/resources/upload и получить URL для закачки".**

1. **Как тестировал 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)  
Проверял, что API правильно логирует ошибки, связанные с некорректными типами данных. С помощью Логирования отслеживал и анализировал проблемы в производственной среде.  
  
Последнюю тестировал - получение списка рекомендованных товаров по предпочтениям**

1. **Как тестировать метод POST, особенности?**

**Особенности метода POST: в отличие от GET-запросов, у которых обычно нет тела, в POST есть. Тело запроса — это данные, которые вы добавляете в базу данных.  
Записывает данные, неидемпотентный, но может быть настроен идемпотентным**

1. **Как тестировать метод GET, особенности?**

**Получить данные, не имеет тела запроса  
GET методы используются для получения информации.   
  
Он используется для получения данных с сервера без их изменения.**

1. **"Как передаём данные с помощью GET**

**С помощью урла и обязательных атрибутов запрашиваются необходимые данные.  
  
Endpoint – это указание на конкретный ресурс, к которому мы хотим получить доступ. Например, если мы хотим получить информацию о пользователях, то endpoint может быть /users.  
  
Query параметры – это параметры, которые мы можем передавать в URL строке после знака “?”. Например, если мы хотим получить список пользователей, отсортированных по имени и возрасту, то URL может выглядеть так: /users?sort=name&age=30. Query параметры позволяют нам фильтровать, сортировать, форматировать данные, которые мы получаем (в основном).**

1. **Как тестировать метод PUT, особенности?**

**Заменить данные или создать  
PUT-запрос всегда заменяет ВСЕ значения во всей записи. Это значит, что если каких-то то значений не хватает — они будут удалены.**

1. **Как тестировать метод Patch, особенности?**

**Обновить часть существующих данных**

1. **Как тестировать метод Delete, особенности?**

**Удалить данные**

1. **Что знаешь про идемподентность? Какие методы идемподентны?**

Идемпотентность

Идемпотентность — это свойство объекта или операции, при повторном применении которой к объекту даётся тот же результат, что и при первом.  
Все, кроме POST  
POST - не идемподентный, тк он каждый раз создает новую запись (если не сделано блока в API на повторную запись с теми же данными

1. **Зачем нужно тестировщику понимать идемподентность конкретной АПИ?**

**Метод POST может быть настроен на проекте идемпотентным, так как того требует бизнес-логика,   
например   
Создание заказа на мойку автомобиля, пока мойка не завершена, при введении тех же данных заказ не будет изменяться и метод не будет создавать новые сущности  
  
Или на некоторых проектах метод POST может заменять метод Delete**

1. **GET может удалить данные?**

**НЕТ, такая ситуация невозможна.   
GET-запрос является методом, который используется для получения данных с сервера.   
Он не предполагает изменения или удаления информации в базе данных. Когда мы отправляем GET-запрос, мы запрашиваем информацию с сервера, и сервер отвечает на этот запрос, предоставляя нужные данные.**

1. **Главное отличия гета от поста**

**Тело запроса**

**POSTMAN**

1. **Как работал с переменными?**

**задавал в переменные: окружение, атрибуты необходимые и переключал**

1. **Какие виды авторизации могут быть? Какие были у тебя?**

**Парольная. Основана на использовании уникального набора символов для каждого пользователя.   
Биометрическая. Основана на уникальных физиологических параметрах или поведенческих характеристиках человека, таких как отпечаток пальца или голосовая идентификация.   
Токенная. Доступ предоставляется с помощью специального устройства, например USB-ключа или смарт-карты.   
OAuth. Протокол авторизации, позволяющий приложениям получать доступ к данным без необходимости предоставления своих учетных данных. Часто используется в интернет-сервисах и социальных сетях.**

1. **Работал ли с коллекциями?**

**Разраб методом круд предеовал коллекцию файликом и я тестил в постмане**

1. **200 статус код это всегда окей ? (ожидаю услышать возможную ошибку в теле ответа)**

**Если в теле ответа есть сообщение об ошибке конкретного приложения, то статус 200 может указывать на то, что сообщение было успешно передано. При условии, что никакие технические проблемы не помешали правильной работе бизнес-логики**

**SWAGGER**

1. **Что делал в SWAGGER?**

**Тестировал API, проверял как отрабатывает запрос, какие обчзательные атрибуты в теле запроса, какие статус-коды в позитивных и негативных проверках, какое тело ответа**

1. **Где были описаны API? В сваггере**

**Клиент-серверное**

1. **Кто такой клиент?**

**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**

1. **Структура URL**

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

**Домен – адрес сайта в интернете**

**Путь – откуда именно берутся данные**

**Строка запроса (состоит из параметров)**

1. **Что такое HTTP протокол?**

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

1. **Из чего состоит http запрос API?**

* **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).**

**Ответ**

* **Версия HTTP-протокола**
* **Статус код**
* **Заголовки**
* **Тело ответа**

1. **HTTP и HTTPS в чем разница? Где можно использовать http?**

**HTTP:  
Не защищает данные, передаваемые между клиентом и сервером.  
Подходит для сайтов, где безопасность данных не критична, например, для открытых ресурсов с общедоступной информацией. можно использовать для внутренних ресурсов, например, портал  
HTTPS:  
Обеспечивает шифрование данных, передаваемых между клиентом и сервером.  
Рекомендуется для сайтов, где требуется защищенная передача данных, например, для интернет-магазинов, банков и других сайтов с конфиденциальной информацией.**

1. **Какие есть виды протоколов?**

**TCP (Transmission Control Protocol): Обеспечивает надежную доставку данных в сети Интернет.  
IP (Internet Protocol): Определяет адресацию и маршрутизацию пакетов данных в сети Интернет.  
  
Протоколы прикладного уровня:  
HTTP (Hypertext Transfer Protocol): Используется для передачи гипертекстовых документов в сети Интернет (например, веб-страниц).  
SMTP (Simple Mail Transfer Protocol): Используется для отправки электронной почты.  
FTP (File Transfer Protocol): Используется для передачи файлов по сети.  
  
Безопасные протоколы:  
HTTPS (Hypertext Transfer Protocol Secure): Обеспечивает защищенную передачу данных по сети Интернет с использованием протокола TLS.  
SSL/TLS (Secure Sockets Layer/Transport Layer Security): Обеспечивает защищенное соединение и шифрование данных.  
  
Управление сетью и протоколы маршрутизации:  
SNMP (Simple Network Management Protocol): Используется для управления сетевыми устройствами и мониторинга их состояния.  
OSPF (Open Shortest Path First): Протокол маршрутизации, используемый для определения оптимального пути в IP-сети.  
  
Протоколы транспортного уровня:  
UDP (User Datagram Protocol): Предоставляет возможность передачи данных без гарантии доставки или управления порядком сообщений.**

1. **Какие методы знаешь?**

**Самые популярные методы API:   
GET – получить данные   
POST – записать данные   
PUT – заменить данные или создать   
DELETE – удалить данные   
+PATCH – обновить часть существующих данных**

1. **Какая архитектура сайта у тебя была? Двузвенная или трехзвенная? Трехзвенная**
2. **Что такое REST архитектура?**

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

**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. **В чем разница REST и SOAP?**

**"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."**

1. **Почему не все используют SOAP?**

**"SOAP API использует встроенные протоколы, что делает его сложным в использовании.**

**В основном используют для Финансовых приложений, требующих точного и строгого представления данных, лучше всего подходит для SOAP. SOAP, основанный на XML, гарантирует единообразное форматирование финансовых транзакций, снижая вероятность ошибок интерпретации данных.**

**Остальные проекты легче реализовать с помощью REST, более гибкий подход"**

1. **В чем разница JSON / XML языков разметки?**

**JSON (англ. JavaScript Object Notation) — текстовый формат обмена данными в REST API.  
Язык разметки.  
Набор пар «ключ — значение»   
Значения могут быть строками, числами, булевыми значениями, объектами, массивами или null   
Функции и даты в обычном JSON не поддерживаются.  
  
XML — для отображения, хранения и транспортировки информации.   
В XML вы изобретаете свои собственные теги, так как в языке нет предопределенных тегов.**

1. **Где используется JSON?**

**Обмен данными в веб-приложениях: JSON часто используется для обмена данными между веб-сервером и клиентским приложением. Например, многие веб-сервисы возвращают данные в формате JSON, который затем обрабатывается клиентским JavaScript кодом.   
  
Хранение конфигурационных данных: JSON может использоваться для хранения конфигурационных данных в формате, который легко читать и записывать как человеку, так и машине.   
  
Хранение данных в базах данных: Некоторые базы данных, такие как MongoDB, могут хранить данные в формате JSON, что делает их удобными для хранения структурированных данных.   
  
API и веб-сервисы: Многие API и веб-сервисы используют JSON для передачи данных между различными системами.   
  
Настройка и обмен данными в микросервисах: JSON часто используется для настройки и обмена данными между микросервисами в архитектуре микросервисов.   
  
Хранение данных в файлах: JSON может использоваться для хранения данных в файлах, таких как настройки приложений или данные для тестирования.   
  
JSON является одним из наиболее популярных форматов обмена данными из-за своей простоты, гибкости и удобства использования для разработчиков.**

1. **Что такое ключ-значение?**

**каждая строка содержит пару "ключ-значение". Например, "name": "John" - ключ "name" с соответствующим значением "John". Ключи в JSON должны быть уникальными в рамках одного объекта. Ключи и значения разделяются двоеточием, а пары ключ-значение разделяются запятыми.**

1. **Какие типы данных поддерживает JSON?**

**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) { "middlename":null }**

1. **В чем разница массива и объекта?**

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" } }

1. **Где может использоваться булеан?**

**BOOLEAN** - хранит логическое булево значение (true/false)

**"Функции и методы: Булевы значения могут быть возвращаемыми значениями функций или методов, чтобы сообщить о результате выполнения определенной операции.**

**Интерфейсы и API: Булевы значения могут использоваться в API или интерфейсах для управления настройками или параметрами.**

**Логика приложений: Булевы значения могут использоваться для управления логикой приложения, например, для включения или отключения определенных функций или возможностей."**

1. **Напиши тело запроса в JSON для передачи «name – Иван»**

{ "name": "Иван" }   
Этот JSON-объект содержит одну пару "ключ-значение", где ключ "name" указывает на значение "Иван". Если требуется передать дополнительные данные, их можно добавить в этот объект.

1. **Как ты понимаешь клиент серверное взаимодействие?**

**Взаимодействие клиента (запрос инициирует), http-протокола, сервера, бд**

1. **Как работает интернет?**

**Интернет — это глобальная сеть, соединяющая миллионы компьютеров и устройств по всему миру. интернет представляет собой сложную систему взаимосвязанных технологий, обеспечивающих обмен данными между устройствами по всему миру.**

1. **Двузвенная/Трёхзвенная архитектура**

**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.**

**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 такоке кеш?**

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. **Какая максимальная длина url? Можно ли туда записать книгу "Война и мир"? 2000-2500 символов**

**"Максимальная длина URL в разных браузерах:**

**Google Chrome — 2 МБ (2 097 152 символа)**

**Mozilla Firefox — неограниченная, но практически после 65 536 символов адресная строка не отображает URL;**

**Edge — 2083 символа, но не более 2048 символов в части пути URL;**

**Opera — неограниченная;**

**Safari — 80 000 символов, после пересечения этого предела страница выдаёт ошибку."**

1. **Какой у тебя Опыт работы и понимание микросервисной архитектуры?**

**"Большая сложная система разбивается на много маленьких независимых блоков**

**Каждая часть такого приложения отвечает за определённую задачу и может быть изменена или расширена без перемен в других. При этом сервисы взаимодействуют между собой с помощью обмена сообщениями "**

1. **Какие методы API кешируются?**

**Кешируются методы, которые получают информацию**

**GET запросы**

**HEAD запросы**

**OPTIONS запросы**

**Виды тестирования**

1. **Какие виды тестирования знаешь?**

**Функциональные (проверяем как работает функционал)   
  
Нефункциональные (определяют характеристики ПО или в нашем случае изменения интерфейса-дизайн, удобство использования, совместимость)   
  
Связанные с изменениями (проверяем как ведет себя система после внесения изменений, т.е. после того, как внесли изменения в код)**

1. **Какие уровни есть в видах тестирования?**

**1.Функциональные виды тестирования: базируются на функциях и особенностях, а также взаимодействии с другими системами**

**- модульное   
- интеграционное Проверяем взаимодействие между компонентами внутри системы; да  
- системное выполняемое на полной, интегрированной системе;**

**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. **В чем разница интеграционных тестов и системных?**

**Интеграционные проверяют связь микросервисов и компонентов,   
  
а системные - проводится тестирование всей системы в целом, включая ее функциональные и нефункциональные возможности и взаимосвязанность разных компонентов и микросервисов.  
  
пример системной: Добавление товара-авторизация-применение промокода-оплата картой-получение письма на почту с номером заказа+ добавление заказа в CRM систему.**

1. **Какие виды тестирования связаны с изменениями?**

**К связанным с изменениями относятся:   
  
Дымовое (Smoke) тестирование - несколько проверок, что система не упала;   
  
Санитарное тестирование - детальное тестирование нашей доработки;   
  
Регрессионное тестирование - проверка, что все доработки спринта не поломали сайт и функционал, который работал до этого.**

1. **В чем разница функционального вида тестирования и санитарного?**

**"Функциональное тестирование проверяет соответствие программного продукта функциональным требованиям, описанным в спецификации или пользовательском сценарии использования.**

**Санитарное тестирование - детальное тестирование нашей доработки;"**

1. **В чем разница между валидацией и верификацией? Что делал ты?**

**Верификация — это процесс проверки, в ходе которого устанавливается соответствие разработанного продукта или системы заранее определенным требованиям, спецификациям или регламентам. В контексте разработки ПО верификация обычно направлена на подтверждение того, что продукт корректно реализован на каждом этапе разработки и что все функциональные и нефункциональные требования выполнены до начала фазы тестирования.  
  
Валидация — это процесс проверки того, что продукт или система соответствует потребностям и ожиданиям пользователей, а также установленным требованиям. В контексте разработки ПО валидация направлена на подтверждение того, что конечный продукт реализует все заявленные функции и способен удовлетворять потребности конечных пользователей.  
  
я с валидацией работал, а как? пример   
код запущен и отрисовывается сайт, верификация дизайн макета**

1. **Зачем нужна пирамида тестирования?**

**Модульные тесты (Unit Tests): Это основа пирамиды. Они проверяют отдельные компоненты или функции кода и должны занимать наибольшую часть тестов. Их легко автоматизировать и они быстро выполняются.**

**Интеграционные тесты (Integration Tests): Проверяют взаимодействие между модулями. Эти тесты занимают средний уровень в пирамиде и меньше по количеству по сравнению с модульными тестами.**

**Системные тесты (System Tests): Проверяют приложение в целом, включая его функциональность и производительность. Их количество меньше, чем интеграционных тестов.  
  
Такая структура помогает эффективно организовать тестирование и обнаруживать дефекты на ранних этапах разработки.**

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;   
  
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.**

1. **Что делал в нагрузочном тестировании? ничего**
2. **Кто проводит модульные тесты? разраб**
3. **Проводил ли ты приемочное тестирование? нет, это делает заказчик**
4. **Что проверял в нефункциональных видах тестирования? Тестирование удобства пользования UX UI | кроссбраузерность**
5. **Связанные с изменениями виды тестирования, расскажи про их последовательность на примере тестирования фичи. дымовое, санитарное, регрессионное**

**Smoke-тесты — это базовые тесты, которые проверяют основные функциональные возможности приложения.   
  
Цель таких тестов — убедиться, что основные возможности системы работают как запланировано.**

**Санитарное - — это узконаправленное тестирование конкретной функции исходя из того, какую задачу вы тестируете. Проверяем, что в рамках задачи все работает согласно заявленным требованиям. Также проверяется связанное окружение.**

**Регрессионное тестирование- — цель регрессионного тестирования: убедиться, что исправление одних багов не стало причиной возникновения других и что внесение изменений в код не создало новых дефектов в уже проверенном коде.**

1. **В какой момент спринта какие виды тестирования проводились?**

**Функциональные виды тестирования проводили всегда, в конце регресс**

1. **Какие методики тестирования ты использовал? (ящики)**

**Черный и серый**

**Black-box (Чёрный ящик)  
При данной стратегии тестировщик осуществляет проверку продукта, не имея информации об особенностях его реализации, используя только интерфейс. Метод имитирует поведение пользователя, у которого нет никаких знаний о внутреннем устройстве программы.**

**Gray-box (Серый ящик)  
Проверка «серого ящика» – это метод тестирования программного продукта или приложения с частичным знанием его внутреннего устройства.**

**Мы знаем примерный алгоритм, архитектуру, внутренние взаимодействия или другие общие описания поведения программы.**

1. **Проводил ли ты бета тестирование? нет**

**SQL**

1. **Чем отличается БД от СУБД?**

**БД представляет собой сами данные, а СУБД - программное обеспечение, которое используется для управления этими данными. СУБД позволяет создавать, изменять, удалять и извлекать данные из базы данных с помощью удобного интерфейса и языка запросов (например, SQL)**

1. **С какими СУБД работал? PostgreSQL**
2. **Какие SELECT операторы ты использовал?**

**SELECT и FROM. Определяют, какие столбцы данных отобразить в результатах.   
GROUP BY. Позволяет объединять строки и агрегировать данные.   
HAVING. Позволяет сортировать данные, которые собираются через GROUP BY.   
ORDER BY. Задает критерии сортировки строк.**

1. **Что делает Distinct, Select top, Count, Avg, like?**

**Distinct в SQL позволяет получить только неповторяющиеся данные из столбцов, игнорируя дублирующиеся значения.   
  
Select top ограничивает итоговый набор записей первыми N строками или первыми N процентами от общего количества строк результата.   
  
Count в SQL считает количество значений указанного столбца.   
  
Avg в SQL вычисляет среднее арифметическое значение множества значений столбца.   
  
LIKE в SQL используется для поиска подстрок. Значения столбца сравниваются с задаваемым после него шаблоном.   
  
Синтаксис всех этих функций имеет общий вид: SELECT AGGREGATE\_FUNCTION(имя столбца) FROM имя таблицы**

1. **Каким селектом убрать дублирующиеся записи?**

**DISTINCT, который удаляет повторяющиеся строки запроса и оставляет только уникальные**

1. **Как написать запрос к таблице users, если я хочу увидеть колонку Name с именем Иван?**

**SELECT Name  
FROM users  
WHERE Name = 'Иван'**

1. **Какие есть джоины?**

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. **По какому принципу происходит объединение таблиц в джоине?**

**Объединение таблиц в SQL-джоне происходит по следующим принципам:   
  
Наличие условия объединения. 3 Для JOIN обязательно условие сопоставления строк исходных таблиц друг с другом, которое представляет собой логическое выражение (предикат).   
Схема таблицы-результата. В неё входят столбцы обеих исходных таблиц (таблиц-операндов), то есть схема результата является «сцеплением» схем операндов.   
Формирование строк таблицы-результата. Каждая строка таблицы-результата является «сцеплением» строки из одной таблицы-операнда со строкой второй таблицы-операнда.   
Определение того, какие именно исходные строки войдут в результат и в каких сочетаниях, зависит от типа операции соединения и от явно заданного условия соединения.**

1. **Как работал с нереляционными базами данных?** **Не работал**
2. **"Реляционная бд отличия от нереляционной**

**"Реляционная база данных (SQL) — база, где данные хранятся в формате таблиц, они строго структурированы и связаны друг с другом. В таблице есть строки и столбцы, каждая строка представляет отдельную запись, а столбец — поле с назначенным ей типом данных. В каждой ячейке информация записана по шаблону.**

**Нереляционная база данных (NoSQL) — хранит данные без четких связей друг с другом и четкой структуры. Вместо структурированных таблиц внутри базы находится множество разнородных документов, в том числе изображения, видео и даже публикации в социальных сетях. В отличие от реляционных БД, NoSQL базы данных не поддерживают запросы SQL."**

1. **Агрегатные функции как работают?**

**Агрегация относится к таким операциям, когда больший набор строк свёртывается в меньший. Типичные агрегатные функции - COUNT, MIN, MAX, SUM и AVG.**

1. **Отличия оператора where vs having**

**WHERE используется для фильтрации записей до группировки, отсеивая те, которые не удовлетворяют заданному критерию. В свою очередь, HAVING применяется после операции GROUP BY и позволяет отфильтровать группы записей по агрегированным значениям, получаемым с помощью функций SUM, AVG или COUNT.**

**DevTools**

1. **Где посмотреть тело, которое передается в апишку POST в консоли? Network**
2. **Что такое токен авторизации, access token и refresh токен.**

**Токен авторизации (Authorization Token) - это строка, которая представляет собой уникальный идентификатор, используемый для аутентификации пользователя и предоставления доступа к ресурсам или функциям веб-приложения или API. Токены авторизации используются для защиты ресурсов от несанкционированного доступа.  
  
Access Token (Токен доступа) - это конкретный тип токена авторизации, который выдается после успешной аутентификации пользователя и предоставляет доступ к защищенным ресурсам или функциям веб-приложения или API. Access Token обычно имеет ограниченное время жизни и может быть использован для аутентификации запросов к защищенным ресурсам в течение этого времени.  
  
Refresh Token (Токен обновления) - это также тип токена авторизации, который выдается вместе с Access Token и используется для обновления Access Token после истечения его срока действия без необходимости повторной аутентификации пользователя. Refresh Token обычно имеет более длительное время жизни, чем Access Token, и служит для продления срока действия доступа к ресурсам.**

1. **В чем разница между авторизацией и аутентификацией?**

**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. **Типы хедеров?**

**Errors. Warnings.**

1. **В каком случае нужно проверять кроссбраузерность?**

**Кроссбраузерное тестирование необходимо проводить, чтобы убедиться, что веб-сайт или веб-приложение выглядит, работает и ведёт себя одинаково или очень схоже в различных браузерах, таких как Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge и других.   
  
Оно направлено на устранение визуальных несоответствий, функциональных проблем и несоответствий в работе с пользователем.   
  
Важность тестирования кроссбраузерности заключается в том, что оно:   
  
— гарантирует согласованный и бесперебойный опыт работы для всех пользователей;   
— расширяет охват рынка;   
— поддерживает доверие;   
— обеспечивает совместимость с мобильными браузерами;   
— влияет на SEO;   
— снижает нагрузку на поддержку и техническое обслуживание.**

1. **Из каких компонентов состоит фронтенд?**

**Фронтенд веб-приложения состоит из нескольких основных компонентов, которые взаимодействуют между собой для отображения пользовательского интерфейса и обеспечения его функциональности. Вот основные компоненты фронтенда:   
  
HTML (HyperText Markup Language): HTML используется для структурирования содержимого веб-страницы. Он определяет элементы и их расположение на странице.   
  
CSS (Cascading Style Sheets): CSS используется для стилизации и оформления внешнего вида веб-страницы. Он определяет цвета, шрифты, отступы, размеры и другие аспекты визуального оформления.   
  
JavaScript: JavaScript используется для добавления интерактивности и динамического поведения на веб-странице. Он позволяет реагировать на действия пользователя, обрабатывать данные и изменять содержимое страницы без необходимости перезагрузки.   
  
Фреймворки и библиотеки: Фреймворки и библиотеки, такие как React, Angular или Vue.js, предоставляют набор инструментов и готовых компонентов для упрощения разработки фронтенда, управления состоянием и создания масштабируемых приложений.   
  
Шаблоны (Templates): Шаблоны позволяют создавать многократно используемые компоненты интерфейса, такие как заголовки, навигационные панели или формы, что упрощает разработку и поддержку кода.   
  
AJAX (Asynchronous JavaScript and XML): AJAX используется для асинхронной передачи данных между браузером и сервером без перезагрузки страницы. Это позволяет создавать более динамичные и отзывчивые веб-приложения.   
  
Responsive Design (Отзывчивый дизайн): Этот подход к дизайну позволяет создавать веб-страницы, которые хорошо отображаются на различных устройствах и экранах, адаптируя свой внешний вид и расположение элементов.   
  
Эти компоненты взаимодействуют друг с другом для создания пользовательского интерфейса, который не только хорошо выглядит, но и обеспечивает удобство использования и функциональность веб-приложения.**

1. **Какие браузеры нужно проверить, когда говорят, что надо проверить все? Chrome, «Яндекс Браузер», Opera, Microsoft Edge, Mozilla, Safari**
2. **Какие браузерные движки знаешь?**

**Google Chrome: Один из самых популярных браузеров среди пользователей по всему миру.   
  
Mozilla Firefox: Еще один широко используемый браузер с открытым исходным кодом.   
  
Apple Safari: Браузер, используемый на устройствах Apple, таких как iPhone и iPad.   
  
Microsoft Edge: Браузер от Microsoft, который поставляется вместо Internet Explorer в Windows 10.   
  
Internet Explorer: В некоторых случаях может потребоваться проверка старых версий Internet Explorer, особенно если ваше приложение должно поддерживать старые версии Windows.   
  
Opera: Альтернативный браузер с небольшой долей рынка, но все же используемый некоторыми пользователями.   
  
Мобильные браузеры: Кроме того, может потребоваться проверка браузеров на мобильных устройствах, таких как Chrome и Safari для мобильных устройств.**

1. **Авторизация, аутентификация, идентификация**

**1. Идентификация (Identification)**

**Идентификация — это процесс определения пользователя или устройства в системе. Это первая стадия взаимодействия с системой, в которой пользователь предоставляет свои учетные данные, чтобы система могла его распознать.**

* **Примеры:**
  + **Ввод имени пользователя или адреса электронной почты.**
  + **Предоставление уникального идентификатора, например, номера сотрудника.**

**2. Аутентификация (Authentication)**

**Аутентификация — это процесс проверки подлинности пользователя, то есть подтверждение того, что он является тем, за кого себя выдает. После идентификации система требует доказательства, что пользователь действительно тот, за кого он себя выдает.**

* **Способы аутентификации:**
  + **Пароль: Ввод пароля, связанного с учетной записью.**
  + **Двухфакторная аутентификация (2FA): Дополнительный уровень проверки, например, отправка кода на телефон или электронную почту.**
  + **Биометрия: Использование отпечатков пальцев или распознавания лица.**

**3. Авторизация (Authorization)**

**Авторизация — это процесс определения прав доступа пользователя к ресурсам системы после успешной аутентификации. Это этап, на котором система решает, какие действия может выполнять идентифицированный и аутентифицированный пользователь.**

* **Примеры:**
  + **Доступ к определенным страницам веб-приложения (например, администраторы могут иметь доступ к настройкам, а обычные пользователи — нет).**
  + **Ограничение доступа к файлам или базе данных на основании ролей пользователей.**

1. **Что делал во вкладке элементы?**

**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).**

1. **Что проверял в CSS?**

**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.)**

1. **Как работал с куками, что делал?**

**Cookies: Отображает все куки, установленные веб-сайтом, включая их значение, время истечения и доступные пути. Работа с куки как добавляются и удаляются.**

**Удаляла кук товара в коризине и корзина обнулялась, чистила куки авторизации (Например до 1 минуты сокращали хранение куки и смотрели удалится ли. )**

1. **Как понять проблема фронта или бэка?**

**Неверное тело запроса и статус-код ошибка = фронт  
 Неверные атрибуты запроса - статус-код ок = бэк  
 Верное тело запроса - статус-код ок = бэк  
 Верное тело запроса - статус-код ошибка = бэк**

**1. Сбор информации о проблеме**

**2. Проверка пользовательского интерфейса (Фронтенд)**

**3. Проверка серверной части (Бэкенд)**

**4. Анализ полученных данных**

**5. Тестирование**

* **Тестирование фронтенда: Проведите тесты, чтобы убедиться, что интерфейс работает корректно с различными вводами и состояниями.**
* **Тестирование бэкенда: Напишите тесты для API, чтобы проверить, возвращает ли он правильные данные и обрабатывает ли запросы корректно.**

1. **Из чего состоит HTTP запрос и ответ?**

**HTTP-запрос:**

1. **Стартовая строка (Метод, URL, Версия протокола)**
2. **Заголовки (Headers)**
3. **Тело запроса (Body) (для методов вроде POST)**

**HTTP-ответ:**

1. **Стартовая строка (Версия протокола, Код состояния, Сообщение о состоянии)**
2. **Заголовки (Headers)**
3. **Тело ответа (Body) (содержит данные, возвращаемые сервером)**

**Эти компоненты вместе обеспечивают полноценное взаимодействие клиента и сервера в интернете.**

**ГИТЛАБ**

1. **Что такое Гитлаб?**

GitLab - это веб-платформа для управления проектами и хранения кода с открытым исходным кодом, которая используется для совместной разработки программного обеспечения.

1. **Для чего используется?**

**Она предоставляет инструменты для хранения репозиториев Git, управления задачами, непрерывной интеграции и развертывания, контроля версий, отслеживания ошибок и других задач разработки. GitLab предлагает облачное и самостоятельное развертывание на серверах пользователя. Он широко используется командами разработчиков для совместной работы над проектами и управления кодом.**

1. **Что такое CI/CD? Что там делать, зачем и как?**

**CI/CD — это одна из DevOps практик, которая позволяет разработчикам чаще и надёжнее развёртывать изменения ПО, свести к минимуму ошибки, повысить темпы сборки и качество разрабатываемого продукта. Представляет собой комбинацию continuous integration и continuous delivery.  
  
CI, или непрерывная интеграция — процесс постоянной разработки ПО с интеграцией в основную ветвь. Автоматически собирает софт, тестирует его и оповещает, если что-то идёт не так.  
  
CD, или непрерывная доставка — процесс постоянной доставки ПО до потребителя. Обеспечивает разработку проекта небольшими частями и гарантирует, что он может быть отдан в релиз в любое время без дополнительных ручных проверок.  
  
Как выглядит цикл CI/CD:  
  
Разработчик пишет код, проводит начальное тестирование, чтобы не было ошибок, и фиксирует изменения в своей рабочей ветке. Затем соединяет модифицированный код с рабочим кодом из основной ветки.  
  
Система, выбранная инструментом CI, замечает, что в коде произошли изменения и запускает автоматическую сборку и автоматическое тестирование программы.  
  
Если автоматическое тестирование прошло успешно, ПО отдаётся для ручного тестирования команде тестировщиков.  
  
После исправления недочётов, обнаруженных во время ручного тестирования, запускается автоматизированная установка ПО на серверах компании.  
  
Осуществляется поддержка новой версии программы и её мониторинг.  
  
Собираются запросы на исправление недочётов и багов, разработчик вносит изменения в код, и процесс повторяется.  
  
Основное преимущество CI/CD в том, что разработчику нужно только написать код, а остальные процессы по тестированию, сборке и доставке проходят автоматически.**

1. **Что такое мерж конфликты? В какой момент происходят?**

**Мерж-конфликты (Merge conflicts) возникают, когда система контроля версий (например, Git) не может автоматически объединить изменения из разных веток (например, из двух веток в Git) из-за конфликта в измененных файлах.  
  
Примеры ситуаций, когда могут возникнуть мерж-конфликты:  
  
Изменения в одной и той же строке: Если два разработчика в разных ветках вносят изменения в одну и ту же строку кода, система контроля версий не может самостоятельно определить, какие изменения следует сохранить.  
  
Удаление или перемещение файлов: Если файл был удален или перемещен в одной ветке, а в другой ветке были внесены изменения в этот файл, возникает конфликт.  
  
Изменения в структуре кода: Изменения, вносимые в структуру проекта (например, переименование переменной или функции), могут привести к конфликту, если они вносятся в разных ветках.  
  
Когда система контроля версий обнаруживает мерж-конфликт, она останавливает процесс слияния и помечает файлы, в которых возникли конфликты. Разработчику необходимо вручную разрешить конфликт, выбрав изменения, которые следует сохранить, и затем продолжить процесс слияния. После разрешения конфликтов изменения объединяются и ветка считается слитой.**

1. **Какие типы веток чаще всего бывают? Какие были у тебя?**

**master/main. Содержит тестированный, стабильный код, который может быть выложен на производство.   
develop. Основная ветка для разработки, содержит актуальный, но не релизный код.   
feature. Ветки для разработки новых функциональностей, которые впоследствии сливаются с develop.   
release. Ветка для подготовки новых релизов, отделяется от develop, затем сливается с master и develop.   
hotfix. Ветки для быстрого исправления ошибок в продакшен-версиях, сливаются с master и develop.**

1. **Как проходил процесс ревью?**

**Разработчик создаёт ветку, вносит изменения и подаёт запрос на слияние этих изменений. В зависимости от уровня доступа разработчика, ветка может быть отправлена либо в копию репозитория, forked на его личный аккаунт, либо напрямую в основной репозиторий.   
После подачи запроса на слияние, ревьюеры могут комментировать или делать предложения. В ответ на обратную связь разработчик может отправить дополнительные коммиты в свою рабочую ветку.   
Ревьюеры могут затем либо объединить, либо закрыть запрос. Пользователи, имеющие право объединить запрос, могут также отправить коммиты в базовую ветку.   
Процесс ревью помогает обнаружить и исправить случайные ошибки и несоответствия.**

1. **Разница монолитного и микросервисного сайта**

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

1. **Как устроен монолитный сайт?**

**Приложение, построенное как цельный продукт с единой кодовой базой.  
  
В монолитной архитектуре все модули и функциональности сильно взаимосвязаны.  
  
Приложение построено как единое целое, где компоненты напоминают звенья замкнутой цепи. Связи между ними настолько сильны, что малейшие изменение будут влиять на работу всего приложения.**

1. **Как устроен микросервисный сайт?**

**Подход, при котором приложение разбивается на компоненты, и каждый из них выполняет определенную функцию, работает автономно.   
  
Сервисы обмениваются данными через API и могут быть разработаны, развернуты и масштабированы независимо друг от друга.**

1. **Как тестировать интеграцию между микросервисами?**
2. **Разница релиза к в монолитный сайт и в микросервисный?**

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

1. **С какой тебе больше нравится работать?**

**микросервисная**

1. **Какие микросервисы у тебя были?**

**создание заказа, отмена заказа, оплата заказа и тд, например, интернет-магазин с тремя микросервисами: «Магазин одежды», «Корзина покупок» и «Оплата»**

1. **Как закатывали фичи на стенды?**

**Релизная ветка создаётся на основе ветки разработки (development). В ней не добавляется новая функциональность, а производится только отладка багов, создание документации и решение других задач, связанных с релизом.   
  
Когда тестировщики скажут, что ветка достаточно стабильная для нового релиза, её сливают в основную ветку (master). Далее на этом коммите создаётся метка (tag), которой присваивается номер версии.**

1. **Сколько стендов было?**

**3 - dev, stage, prod**

1. **Кто собирал релизную ветку?**

**собирает команда разработчиков, которая готовит выпуск новой версии проекта**

1. **Что такое мастер ветка, мерж, комит, деплой,**

**Мастер-ветка (иногда её называют main) — это главная ветка проекта, в которую заливается только рабочий проверенный код. В этой ветке находится тот же самый код, что и на боевом сайте.   
  
Мерж (слияние веток) — это перенос кода из одной ветки в другую. Например, когда разработчик заканчивает работу над веткой, например, сделал новый функционал или поправил багу, он сливает её в мастер. В мастере код проверяется ещё раз и выкладывается на боевой сервер.   
  
Коммит — это отправка правок обратно в репозиторий. Чтобы правки увидели остальные, можно коммитить и один файл, и сразу несколько.   
  
Деплой не упоминается напрямую, но может упоминаться в контексте ветки master, когда в ней обычно хранится приложение с полным набором функций и дизайном, готовое к деплою (то есть публикации в App Store или загрузке на сервер).**

1. **В каком случае ты идешь к девопсу**

**если деплой не прошел**

1. **Работал ли с Дженкинсом?**

**да, там автотесты у нас были**

**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.

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. **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. **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.**

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

**Эррор или дебаг - например, обратился пользователь, что оплатил, а оплатане 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,**

**Мобилки**

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.**

1. **Как и где качал сборки?**

**Для Айфон Через приложение Test flight**

**Для Андроид Чат бот APK сборки Андроида.**

1. **Как устанаваливаются ios/android приложения?**

**для Android — это файл с расширением .apk (Android Application Package).  
  
Он является архивом, содержащим все ресурсы и файлы, необходимые для установки и запуска приложения на устройствах под управлением операционной системы Android.  
  
Внутри архива содержатся:  
  
— файлы AndroidManifest.xml (содержит информацию о приложении и его конфигурации),  
— .dex (файлы байт-кода Dalvik),  
— ресурсы (изображения, макеты, строки и т. д.),  
— исполняемые файлы и библиотеки.  
  
.ipa (от iOS App Store Package) — формат архивных файлов приложений от Apple для iPhone, iPod Touch и iPad.  
  
Файлы с данным расширением хранятся в магазине App Store и загружаются с помощью iTunes для использования на этих устройствах. Как правило, файлы этого формата шифруются с применением технологии Apple’s FairPlay DRM.  
  
Каждый .ipa-файл — это бинарный файл для ARM-архитектуры, который на деле является сжатой в формате ZIP-папкой определённой структуры и содержит исполняемый файл, файлы ресурсов, такие как видео, аудио и изображения.  
  
Файл .ipa может быть установлен только на iPhone, iPod Touch, iPad, компьютерах Mac с процессором Apple.**

1. **Работал ли с андроид студио и что делал? да**
2. **Какие снифферы использовал?**

Charles Proxy, XCode и Android Studio.

1. **Снимал логи с Эмулятора в андройд студии? да**
2. **Работал ли с Дженкинсом? да**
3. **Как связать телефон со сниффером на пк**

* Настройка сети
* Настройка сниффера
* Работа с HTTPS-трафиком
* Анализ и фильтрация трафика

1. **Функционал сниффера и что это такое вообще**

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