ОТВЕТЫ НА ТЕСТОВОЕ ЗАДАНИЕ

1. Почему вы хотели бы работать тестировщиком ?

Мне всегда было интересно находить нестандартные пути использования программ. Я часто сталкивался с багами в повседневной работе и начал интересоваться, как можно системно проверять продукты. Постепенно изучил основы тестирования, применял эти навыки в коммерческой деятельности, и понял, что мне подходит этот путь: он требует внимания к деталям, логики и ответственности - именно то, что мне близко.

2. Какие проблемы может выявить тестирование веб-приложения?

1) Некорректная вёрстка на разных устройствах/браузерах (кроссбраузерность).
2) Ошибки валидации пользовательского ввода.
3) Неработающие ссылки или кнопки.
4) Утечка данных / security issues.
5) Неверная логика (неверное отображение цен, расчётов).
6) Ошибки 404/500, 3xx - некорректные редиректы.
7) Проблемы производительности и времени загрузки.
8) Ошибки взаимодействия с базой данных/API.
9) Проблемы с сессиями, авторизацией.
10) Ошибки UI/UX: недоступность, плохая навигация, неочевидные элементы.

3. Какие коды ответов HTTP вы знаете ?

Выделяют несколько основных HTTP-кодов по категориям:

1) **2xx** - всё хорошо, запрос выполнен.

- **200 OK** - запрос успешно выполнен (данные получены или результат применён).
- **201 Created** - ресурс создан (например, при POST-запросе).
- **204 No Content** - запрос был успешен, но содержимого нет (например, при удалении).

2) **3xx** - нужен дополнительный шаг, например, follow redirect.

- **301 Moved Permanently** - ресурс окончательно перенесён, используйте новый URL.
- **302 Found** - временное перенаправление.
- **304 Not Modified** - ресурса не изменился, браузеру можно использовать.

3) **4xx** - проблема в запросе со стороны клиента.

- **400 Bad Request** - неверный запрос, плохой формат.
- 4**01 Unauthorized** - нужна авторизация.
- 4**03 Forbidden** - доступ запрещён, даже если авторизован .
- 4**04 Not Found** - ресурс не найден.

4) **5xx** - проблема со стороны сервера.

- **500 Internal Server Error** - на сервере произошла непредвиденная ошибка .
- **502 Bad Gateway** - сервер-посредник вернул некорректный ответ.
- **503 Service Unavailable** - сервер временно перегружен или на техобслуживании.

4. Приоритеты.

Сценарий 1.

1) Уточнить у разработчиков актуальные оценки по задачам.
2) Пересчитать план: что можно успеть, что нет.
3) Подготовить список приоритетных задач MVP (минимально жизнеспособной версии).
4) Созвониться/написать клиенту: объяснить риски и предложить:перенести срокили снизить объем работ (минимальный функционал).
5) Подтвердить финальное решение документально, во избежание непониманий в будущем.

Сценарий 2.

1) Уведомить руководителя / тимлида.
2) Сравнить приоритетность: какой проект важнее, прежде всего для нашей компании.

3) Проверить, можно ли делегировать один из проект коллеге.
4) Если нельзя - предупредить одного из клиентов и предложить альтернативу (например, предложить перенести на другое время или на другой день в связи с уже расписанным корпоративным графиком).

5. Описание проблем.

Сценарий 1: Одна из лампочек в квартире перегорела.

Описание проблемы:
В комнате одна из лампочек перестала включаться при подаче питания, несмотря на то, что другие лампы и электроприборы работают исправно.

Предусловие:

- Светильник в зале установлен и подключён к электричеству.
- В щитке включен автомат/предохранитель, питающий этот светильник.
- Все остальные лампы и устройства в квартире работают нормально.

Шаги воспроизведения:

1) Подойти к выключателю/выключателям, управляющим светом в зале.
2) Переключить соответствующий выключатель в положение «включено».

3) Обратить внимание на светильник: одна лампа не светится, несмотря на подачу питания.

Ожидаемый результат:
При включении света в зале все лампы светильника должны загореться. Свет ровный, без мигания и перебоев.

Фактический результат:
Одна лампочка остаётся тёмной, хотя питание явно подаётся (свет от других ламп работает как обычно).

Приложения:
Фото или видео светильника с одной потухшей лампой при включённом свете.

Сценарий 2.

1) Задокументировать нюансы:

- Указать, что ошибка появляется не каждый раз - тэг «интермиттирующий» в заголовке и описании.
- Тщательно указать всю известную информацию: браузер, версия, ОС, окружение, частота появления (например, «2 из 10 запусков»)
- Прикрепить всё, что есть: логи, скриншоты, видео, дампы - даже если это частичная информация, HAR-файл.

2) Повторить попытки:

- Запустить тест многократно (рекомендуется ≥ 10-15 раз), чтобы попытаться поймать ошибку повторно.

- Настроить автоматизированный скрипт или ручной цикл для стабильного запуска.

3) Изолировать условия:

- Тестировать в разных средах: dev, staging, prod-клонах, чтобы понять влияние окружения.
- Переключаться между браузерами, устройствами, сетевыми условиями - поискать закономерности.

4) Добавить логирование и отладку

- Добавить подробные логи (debug/trace) вокруг подозрительных мест.
- Использовать брейкпоинты, профайлер, трассировки - всё, что позволит отследить состояние системы при проявлении бага .

5) Анализировать и варьировать:

- Анализировать логи: искать повторяющиеся шаблоны, совпадения времени или событий .
- Варьировать параметры: входные данные, тайминги, нагрузку - чтобы увеличить шанс воспроизведения .

6) Сравнить код и окружение

- Ознакомиться с участками кода, которые вероятно вызывают эту проблему.
- Сопоставить с документацией и спецификацией - возможно, что ранее уже правили поведение, но код устарел.

7) Вовлечь команду:

- Поделиться данными (логи, записи, статистика) с разработчиками - вместе найти причину.
- Организовать совместный разбор: тестировщик + дев - с шоу-кейсом и логами - лучше понять проблему.
- Подключить мониторинг / алерты на проде, чтобы поймать баг снова и вовремя получить данные.

6. Agile vs Waterfall.

Для меня ближе Agile. Он даёт гибкость, обратную связь, возможность быстро внести изменения. В том числе был опыт работы в Agile, могу назвать его определённо положительным. В Waterfall всё поэтапно и строго, но часто в конце понимаешь, что требования устарели. Agile позволяет работать итерационно, улучшать продукт постепенно, что особенно важно при работе с UI/UX.

В плане именно эффективности тестирования, Agile позволяет начинать работать с дефектами намного раньше, чем Waterfall.

Тестирование начиная с ранних этапов SDLC обеспечивает:

1) Снижение затрат – ранние дефекты исправлять дешевле: устранение на этапе требований или дизайна примерно в 100 раз дешевле, чем в продакшене.

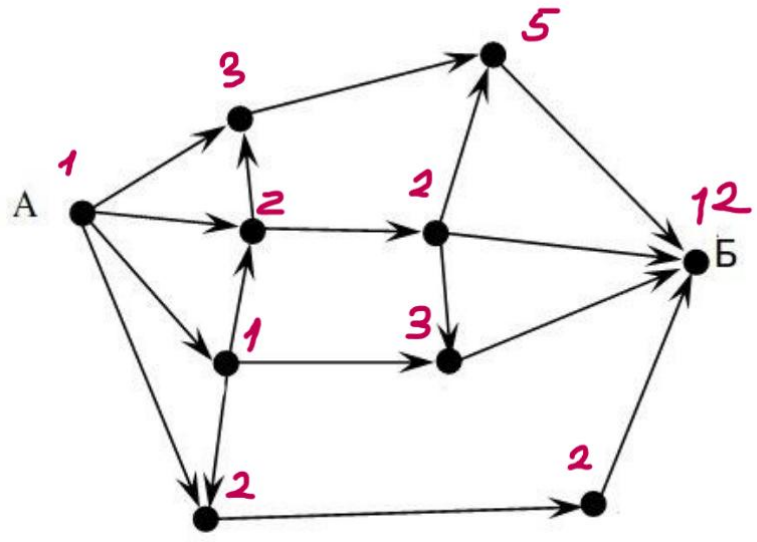
2) Ранняя диагностика – тестирование на этапах юнит-/интеграционных тестов позволяет находить ошибки раньше, когда они ещё мелкие и простые в исправлении.

3) Ускорение выпуска – выявление и устранение багов заранее устраняет узкие места в конце SDLC.

4) Рост качества продукции – постоянные обратные связи (CI/CD) и автоматизация обеспечивают стабильно высокое качество продукта.
5) Повышение эффективности команды – тестировщики, разработчики и аналитики работают вместе с самого начала, сокращая коммуникационные задержки.
6) Непрерывное тестирование и автоматизация – тесты запускаются на каждом коммите, поддерживая качество и предотвращая накопление технического долга.
7) Проактивный подход – акцент на профилактике дефектов через TDD/BDD, ревью требований и статический анализ.
8) Более надёжная архитектура – ранние тесты способствуют построению модульной и тестируемой архитектуры.
9) Безопасность и нефункциональные требования – интеграция тестирования безопасности, производительности и UX с самого начала позволяет избежать проблем позже.
10) Высокая удовлетворённость заказчиков – быстрее релизы с меньшим количеством багов ведут к повышению доверия пользователей и бизнеса .

7. Сколько путей из точки А в точку Б.

В данном ориентированном графе можно выделить 12 возможных путей. На изображении ниже, цифрами подписано количество путей из точки А до каждой из точек. Для первых точек каждого ряда они находятся визуально, для дальнейших сложением числа всех путей от прошлых граничных точек.



8. Три головы Змея Горыныча.

Для решения данной задачи я решил, что наиболее рациональным подходом будет создать небольшой Python скрипт. В нем я оставил небольшие комментарии, для описания этапов. Ответ: 15 минут.

```
# Определяем стороны в которые смотрят головы
head1 = ["forward", "back", "left", "right"]
head2 = ["back", "left", "right"]
head3 = ["right", "left", "forward"]

# Продолжительность каждой фазы
dur1, dur2, dur3 = 10, 15, 20
total = 180
# Определение направления каждой головы в минуту t
count = 0
for t in range(total):
    d1 = head1[(t // dur1) % len(head1)]
    d2 = head2[(t // dur2) % len(head2)]
```

```
    d3 = head3[(t // dur3) % len(head3)]
    if d1 == d2 == d3:
        count += 1

print("Всего минут, когда все три головы смотрели в одну сторону:", count)
```