1.Где передаются параметры в GET-запросе?

Параметры в GET-запросе передаются в URL (Uniform Resource Locator) после знака вопроса "?". Этот формат запроса часто используется при запросах веб-страниц и данных из веб-сервера. Параметры передаются в виде пар "ключ=значение", разделенных символом амперсанда "&".

Например:

https://www.example.com/page?name=John&age=30

В данном примере,

URL https://www.example.com/page - это адрес страницы, а параметры name и age передаются в GET-запросе и содержат значения "John" и "30" соответственно.
Сервер может обработать эти параметры и вернуть страницу с учетом переданных данных.
Клиенты (например, браузеры) могут отправлять GET-запросы, добавляя параметры к URL, что позволяет передавать информацию на сервер для обработки.

2.Где передаются параметры в POST-запросе?

Параметры в POST-запросе передаются в теле HTTP-запроса, а не в URL, как в случае GET-запросов. POST-запросы обычно используются для отправки данных на сервер, например, при отправке форм на веб-страницах или при выполнении других операций, которые требуют передачи данных на сервер.
Данные в POST-запросе передаются в теле запроса в виде пар "ключ=значение".

Типичный заголовок запроса выглядит следующим образом:

POST /example-endpoint HTTP/1.1
Host: www.emple.com
ContentPOST /example-endpoint HTTP/1.1 Host: www.example.com Content-Type: application/x-www-form-urlencoded Content-Length: 29
name=John&age=30&city=New+York-Type: application/x-www-form-urlencoded
Content-Length: 2
namehn&age=3

В этом примере, данные name=John&age=30&city=New+York передаются в теле POST-запроса.
Заголовок Content-Type: application/x-www-form-urlencoded указывает на то, что данные передаются в формате URL-кодирования, который является одним из распространенных способов передачи данных в POST-запросах.
На сервере обработчик POST-запроса может извлечь параметры из тела запроса и выполнить необходимую обработку данных.

3.Может ли успешно пройти такой запрос? Если нет то почему?

```
curl--request GET \
 --url http://example.com:800/table_order/ \
 --header 'Content-Type: application/json' \
--data
 '{
"operation": "paid",
"table": 19,
 "waiter": 1111
 }'
```

Запрос имеет несоответствие между методом HTTP-запроса и заголовками/данными, что может вызвать проблемы при выполнении.
Указан метод GET с помощью --request, но при этом используете заголовок Content-Type: application/json и передаете данные в формате JSON с помощью --data.

GET-запросы обычно не содержат тела с данными, и заголовок Content-Type с типом application/json не является стандартным для GET-запросов.

Обычно параметры GET-запроса передаются через URL.
Для того, чтобы передать данные, в этом примере, лучше использовать метод POST или PUT вместо GET.

Вот исправленный запрос с использованием метода POST:

```
curl --request POST \
--url http://example.com:800/table_order/ \
 --header 'Content-Type: application/json' \
--data
 '{
 "operation": "paid",
"table": 19,
"waiter": 1111
}'
```

4.Может ли успешно пройти такой запрос? Если нет то почему?

```
curl--request Post \
--url http://example.com:8000/table_order/ \
--header 'Content-Type: application/json' \
--data '{
"CheckId": "71847473-f289-4582-bbfc-138d5596fd56",
 "Pin" : "99995", // Пин сотрудника
"Name" : Sergey,
"Register": true,
"Amount": 100,
"Payment",
"DiscountId": "248c2b71-56fd-4a04-81ce-cbe0acadesf9", // SUROK - OK
 }'
```

В этом запросе есть несколько синтаксических и структурных ошибок.
Вот исправленный запрос с комментариями о том, что было изменено:

```
curl --request POST \
 --url http://example.com:8000/table_order/ \
--header 'Content-Type: application/json' \
--data
'{
"CheckId": "71847473-f289-4582-bbfc-138d5596fd56",
 "Pin" : "99995",
 "Name" : "Sergey",
"Register": true,
 "Amount": 100,
 "Payment": "",
 "DiscountId": "248c2b71-56fd-4a04-81ce-cbe0acadesf9"
}'
```

Изменения, которые были внесены:

- Исправлено --request Post на --request POST (POST с большой буквы).
- Удалены лишние комментарии и символы //.
- Добавлены кавычки вокруг имени "Sergey".
- Добавлено значение для ключа "Payment" ("Payment": "").
- Удалена запятая после "DiscountId".

Теперь запрос должен быть корректным с точки зрения синтаксиса. Однако, чтобы запрос успешно прошел, также потребуется убедиться, что указанный URL (http://example.com:8000/table_order/) существует и обрабатывает POST-запросы с данными в формате JSON.

5.Почему в предыдущем запросе "99995" в кавычках, а 100 без, если и то и то цифры?

"99995" указано в кавычках, а 100 без них.
Это связано с тем, что в формате JSON значения должны быть заключены в двойные кавычки, а не-строковые значения, такие как числа, булевы значения или null, не требуют кавычек.

Итак, "99995" указано в кавычках, потому что это строковое значение, а 100 - это числовое значение и не требует кавычек.

```
{
"CheckId": "71847473-f289-4582-bbfc-138d5596fd56",
"Pin" : "99995", // строка - в кавычках
"Name" : "Sergey", // строка - в кавычках
"Register": true, // булево значение - без кавычек
"Amount": 100, // числовое значение - без кавычек
 "Payment": null, // null - без кавычек
 "DiscountId": "248c2b71-56fd-4a04-81ce-cbe0acadesf9" // строка - в кавычках }
```

6. Дана sql таблица. Есть ли разница между ячейками со значением '' (визуально отображаются как пустые) и ячейками со значением null. Если да то какая?

Да, есть разница между ячейками со значением '' (пустая строка) и ячейками со значением NULL в SQL.

'' (пустая строка): Это строка, которая фактически содержит ноль символов, но она все равно является строкой. Визуально она может отображаться как пустое поле, но она имеет значение и считается не пустой, так как это строка с нулевой длиной.

NULL: Это специальное значение в SQL, которое указывает на отсутствие значения или отсутствие данных в ячейке. Это не является строкой, числом или любым другим конкретным типом данных, а скорее означает, что данные отсутствуют.

Разница важна при выполнении операций с данными. Например, при сравнении значений:

- Если сравниваете значение '' с какой-то строкой, то будет выполнено сравнение строк. Например, 'abc' = '' вернет false.
- Если сравниваете значение NULL с какой-то строкой, то результат будет неопределенным (не true, не false), так как NULL означает отсутствие значения.
- При выполнении операций сравнения (например, IS NULL или IS NOT NULL) можно проверить, является ли значение NULL, но можно использовать такие операции для проверки пустой строки ''.

7.Какой запрос сложнее (вызывает бОльшую нагрузку на БД) и почему?

```
SELECT example, COUNT (*)
FROM tablename
WHERE example IN ('abc', 'def)
GROUP BY example
ORDER BY example
```
или
```
SELECT example, COUNT (*)
FROM tablename
GROUP BY example
HAVING example IN ('abc', 'def)
ORDER BY example
```

Два SQL-запроса выполняют примерно одни и те же операции, и разница в нагрузке на базу данных будет незначительной. Однако есть небольшие различия в их логике, которые могут повлиять на производительность в зависимости от объема данных и индексации таблицы:

Первый запрос:
Этот запрос фильтрует строки таблицы tablename, оставляя только те, в которых значение столбца example равно 'abc' или 'def'. Затем он выполняет группировку и подсчет количества строк в каждой группе. И, наконец, сортирует результаты по значению example.

Второй запрос:
В этом запросе сначала выполняется группировка всех строк таблицы tablename по значению example. Затем с помощью HAVING происходит фильтрация результатов так, чтобы были включены только те группы, у которых значение example равно 'abc' или 'def'. Наконец, результаты сортируются по значению example.

С учетом объема данных и индексации:

- Если таблица tablename имеет большой объем данных, и индекс на столбце example существует, то первый запрос, который сначала фильтрует данные, может быть немного более эффективным, так как он уменьшает объем данных для группировки и подсчета.
- Второй запрос, который сначала выполняет группировку и затем фильтрацию с помощью HAVING, может потребовать больше вычислительных ресурсов при группировке всех строк, включая те, которые не участвуют в итоговом результате.

Однако разница в производительности между этими двумя запросами может быть минимальной, и это может зависеть от конкретной реализации БД, объема данных и индексации. Важно профилировать запросы и анализировать выполнение на конкретной базе данных, чтобы точно определить, какой из них более эффективен для вашего случая.

8.Представим приложение. ТЗ на него такое: На экране отображается текст "Введите ваш возраст", поле для ввода возраста, кнопка отправки данных. Когда пользователь вводит свой возраст и отправляет данные, на экране отображается текст "Поздравляем! Вы прожили n лет".
Какие классы эквивалентности можешь выделить для тестирования этой программы?

Для тестирования данного приложения можно выделить следующие классы эквивалентности:

Корректный ввод возраста:
- Ввод корректного возраста (например, от 1 до 150).
- Ввод минимально допустимого возраста.
- Ввод максимально допустимого возраста.
- Ввод возраста, находящегося в разумных пределах (например, от 18 до 100).

Некорректный ввод возраста:
- Ввод отрицательного значения.
- Ввод нуля.
- Ввод возраста, который превышает разумные пределы (например, очень большое число).
- Ввод текстовой строки вместо числа.

Отправка данных:
- Нажатие кнопки отправки данных после ввода корректного возраста.
- Нажатие кнопки отправки данных после ввода некорректного возраста.

Отображение результата:
- Проверка, что после отправки корректного возраста отображается сообщение о количестве прожитых лет.
- Проверка, что после отправки некорректного возраста отображается сообщение об ошибке или предупреждение.

Работа с пустым полем ввода:
- Отправка данных без ввода возраста.
- Проверка, что при отправке данных без ввода возраста отображается сообщение о необходимости ввести возраст.

Повторное взаимодействие:
- Повторное ввод корректного возраста и отправка данных после первой отправки.
- Повторное ввод некорректного возраста и отправка данных после первой отправки.
- Проверка, что результат обновляется при повторной отправке данных с новым возрастом.

Работа с разными языками:
- Проверка, что приложение корректно работает с разными языковыми настройками (если поддерживает мультиязычность).

9.Код-ревью это статическое или динамическое тестирование? Или это не тестирование?

Код-ревью не является тестированием. Это процесс статической проверки кода, в ходе которого один или несколько разработчиков анализируют код, написанный другими разработчиками, с целью выявления ошибок, улучшения качества кода, соблюдения стандартов и передачи знаний.
Важные моменты отличия код-ревью от тестирования:

Цель: Код-ревью направлено на обнаружение проблем в коде и улучшение его качества, а также обмен знаниями и передачу опыта между разработчиками. Тестирование, напротив, оценивает работоспособность программы и выявляет дефекты, ошибки и недоработки.

Время выполнения: Код-ревью производится до того, как код попадет в продакшн (на этапе разработки), в то время как тестирование проводится после завершения кода (на этапе тестирования).

Методика: Код-ревью включает в себя анализ кода, его структуры, логики и стандартов написания кода, а также проверку на соответствие требованиям и спецификациям. Тестирование, с другой стороны, выполняет тесты на исполнимом коде, чтобы проверить его работоспособность.

Участники: В код-ревью обычно участвуют разработчики и инженеры, в то время как тестирование чаще всего проводится тестировщиками или автоматическими средствами тестирования.

Хотя код-ревью и тестирование имеют разные цели и методы, они дополняют друг друга в обеспечении качества программного обеспечения. Код-ревью может помочь выявить проблемы раньше, чем они достигнут этапа тестирования, что уменьшает затраты на исправление ошибок в более поздних стадиях разработки.

10. Предположим, ты работаешь некой компании и у вас есть мобильное приложение. От пользователя пришла обратная связь с багом. Он прислал видео, где видно все его шаги. Ты проделал все то же самое, но баг не воспроизводится. Что будешь делать?

Следует принимать следующие шаги:

Анализ видео и описания пользователя:
Внимательно изучить видео и описание пользователя. Обратить внимание на каждый шаг, который пользователь выполнил, и удостовериться, что я выполнили точно такие же действия. Может быть, в видео есть детали, которые я упустил.

Сравнение окружения:
Проверить, насколько схожи окружения, в которых тестируется приложение. Это может включать в себя версию операционной системы, разрешение экрана,

наличие установленных сторонних приложений и так далее. Иногда баги могут зависеть от конкретных условий окружения.

Использование разных устройств:

Если есть доступ к нескольким устройствам, попробовать воспроизвести баг на других устройствах. Некоторые баги могут быть устройствоспецифичными.

Проверка на разных версиях приложения:

Убедиться, что у меня и у пользователя установлены одинаковые версии приложения. Иногда баги могут быть связаны с конкретной версией приложения.

Логирование и мониторинг:

Если есть доступ к логам приложения, активировать и и попробовать воспроизвести баг еще раз. Логи могут предоставить дополнительную информацию о том, что происходит внутри приложения в момент возникновения бага.

Обратная связь с пользователем:

Поддерживать открытый канал общения с пользователем, который сообщил о баге. Спрашивать у него дополнительные детали, может быть, уточнить, какой тип устройства и операционную систему он использует, и стараться максимально понять его сценарий использования.

Тестирование на реальном устройстве пользователя:

Если это возможно и безопасно с точки зрения безопасности данных, попросить пользователя установить специальную версию приложения с дополнительными инструментами для логирования и отладки. Это может помочь собрать дополнительные данные для выявления и устранения бага.

Обновление приложения:

Проверить, были ли выпущены обновления приложения после того, как пользователь обнаружил баг. Возможно, баг был исправлен в новой версии, и можно предложить пользователю обновить приложение.

Документация и отчеты об ошибках:

Поддерживать детальную документацию о багах и отчеты об ошибках. Это поможет в дальнейшем отслеживать и анализировать подобные инциденты и баги.