Письменные ответы на вопросы

1. Чтобы составить тест-кейсы для этой задачи я бы использовал технику тест-дизайна ТПР и попарного тестирования.
   Это обусловлено тем, что поиск квартиры для покупки имеет много параметров, каждый из параметров может принимать несколько значений.
   Финальное количество проверок после составления таблицы принятия решений можно рассчитать по формуле N = n1 * n2 * n3 * n4 * n5. После вычисления получится 360 проверок (2*5*9*2*2).
   Эти проверки можно оптимизировать, для этого я бы воспользовался техникой попарного тестирования и получил бы в финале 61 проверку.

2. В данном тест-кейсе я бы изменил:

   - **Заголовок**. Из указанного заголовка не очень понятно, что именно проверяют этим тест-кейсом. Также он не соответствует структуре наименования ТК (что проверяем, в какой части приложения, при каких условиях).
     Если этой проверкой проверяется вид главной страницы для неавторизованного пользователя, я бы составил заголовок например так: “Отображение кнопки “Вход/регистрация” на главной странице, если пользователь не авторизован”. Или *“Отображение главной страницы с ограниченным функционалом для неавторизованного пользователя”* — в этом случае внутри необходимо конкретизировать ограничения внутри ТК.

   - **Шаги воспроизведения.** Я бы написал немного короче и использовал бы глаголы в инфинитиве. Например: *“Перейти на страницу https://practicum.yandex.ru/”* или *“Открыть https://practicum.yandex.ru/”*

   - **Ожидаемый результат**. Из указанного результата не совсем понятно как должно себя повести приложение. Я бы по пунктам конкретизировал, каким должен быть ожидаемый результат — какие должны быть ограничения функционала / отличия для неавторизованного пользователя. Например: *“1. Открылась главная страница Яндекс.Практикум 2. Отображается кнопка “Войти””*

- **Уникальный номер.** У каждого тест-кейса должен быть свой ID. Если в этом случае он отсутствует — его следует добавить.

3. Так как архитектура приложения состоит из трёх частей, выявленная ошибка может быть в любой из них. Для того, чтобы локализовать баг, можно использовать Devtools. С его помощью можно выяснить, какой запрос отправляется на бэкенд и тем самым исключить или утвердить, что ошибка на фронтенде. Нужно посмотреть во вкладку “Сеть” — при включенном чек-боксе, его параметр в теле запроса должен иметь значение True. Если это не так — значит, что формируется некорректное тело запроса. Соответственно, ошибка где-то на стороне фронтенда. В другом случае ошибка либо на стороне бэкенда или базы данных, ибо фронтенд посылает правильный запрос.

4. Обычно GET-запрос используется для получения информации с сервера без её изменений. Таким образом в тривиальной ситуации выполнение такого запроса не должно приводить к удалению или модификации информации в базе данных. Для этих целей принято использовать другие типы запросов, например, PUT, POST или DELETE запросы.
   Однако, как мне кажется, возможны исключения, например, если сервер неправильно обрабатывает GET-запрос или если разработчик сам прописал в коде такое действие.
   Тем не менее, GET-запрос предназначен только для чтения, любое другое поведение является ошибкой.

5. Для таблицы **employee**: Первичный ключ (PK) — id (идентификатор сотрудника). У каждого сотрудника должен быть свой уникальный номер. Внешний ключ (FK) — position_id (идентификатор должности). Он отсылает к таблице position.
   Для таблицы **position** первичным ключом будет id (идентификатор должности). У каждой должности также должен быть свой уникальный номер.

6. **Запрос**:
   ```
   SELECT e.fio AS employee_fio,
          p.name AS position_name,
          p.salary AS salary
   FROM employee AS e
   INNER JOIN position AS p ON e.position_id = p.id
   ```

7. В этом коде не хватает оператора, который будет сравнивать “a” с результатом деления — ожидаемый и фактический результаты.
   Верный код:
   ```
   def test_integer_division():
      a = 5//2;
      assert a == 2

   test_integer_division();
   ```

8. КЭ — это набор данных, которые обрабатываются программой по одному алгоритму или приводят к одному результату. Такие классы могут быть двух видов:
   Диапазон: интервал значений. В этом случае, как мне кажется, КЭ и ГЗ существовать по отдельности не могут, ибо очень важно проверять границы этих интервалов. Например, проверка, что пользователь совершеннолетний. В этом случае есть 2 Класса Эквивалентности: 0-17 и 18 - ∞. Очень важно проверить значения на границе этих диапазонов.
   Набор: объекты, объединённые одним свойством. В этом случае нет ни интервалов значений, ни их границ. Соответственно, КЭ и ГЗ могут существовать по отдельности. *Например:* “Тип значений: буквы русского алфавита”

9. Как мне кажется, исключать проверки в середине диапазона не стоит. Несмотря на то, что именно на границах чаще возникают ошибки, значения в середине диапазонов и на их границах проверяют разные нюансы. Значениями в середине диапазона проверяется корректность работы приложения внутри диапазона. Значениями на границах же можно выявить ошибки работы программы при смене диапазонов. Например, если в коде могут быть перепутаны знаки больше, меньше или равно. Выполнение всех необходимых проверок делает продукт более качественным.
   Однако ситуации могут быть разными, например сроки могут быть очень сжатыми, людей может не хватать, а объем проверок может быть очень большим. Думаю, что в нетривиальной ситуации и по согласованию с коллегами можно исключить проверки в середине диапазонов в пользу их границ.

10. *Для того, чтобы прислать логи разработчику нужно:*

Мобильная версия на ANDROID
Можно воспользоваться Logcat в Android Studio или вывести логи в терминал: подключить телефон по USB, перевести его в режим разработчика, открыть Logcat, воспроизвести ошибку, найти лог с помощью фильтра, сохранить лог в файл и отправить его разработчику

Мобильное устройство на iOS
Воспользоваться инструментом iMazing: подключить телефон к компьютеру по USB, воспроизвести баг, сохранить логи в файл и отправить его разработчику

Отчёт о тестировании

Функциональное тестирование веб-приложения

Приложение проверено на стенде
https://e2d10db8-43ca-4925-b2b7-c44bddeed7c9.serverhub.praktikum-services.ru/order .
Все известные требования были покрыты чек-листом:
ЧЛ Сделать заказ Яндекс самокат Гончаров Сергей QA+

Результаты выполнения тестов можно посмотреть здесь:
ЧЛ Сделать заказ Яндекс самокат Гончаров Сергей QA+ .

Из **164** проверок в Google Chrome успешно прошло во всех разрешениях **139**, а в Браузере Яндекс во всех разрешениях экрана успешно прошло **141**. Не прошло: Google Chrome — **25**, Яндекс Браузер — **23**.

Список багов, найденных при тестировании, разбит по приоритетам:

1. Блокирующие:

- Нет

2. Критичные:

- Баг 1
- Баг 2

3. Средний приоритет:

- Баг 1

- Баг 2
- Баг 3
- Баг 4
- Баг 5
- Баг 6
- Баг 7
- Баг 8
- Баг 9
- Баг 10
- Баг 11
- Баг 12
- Баг 13
- Баг 14
- Баг 15
- Баг 16
- Баг 17
- Баг 18
- Баг 19

4. Низкий приоритет:

- Баг 1
- Баг 2
- Баг 3

Заключение:

1. ***Какой баг показался самым критичным?***

   Невозможно оформить заказ используя Google Chrome

2. ***На твой взгляд, какая самая «хитрая» серая зона есть в требованиях?***

- Нет подробного описания взаимодействия с календарём. Например, можно ли сделать заказ на сто лет вперёд?
- Нужно ли окно “Подтверждения заказ” после клика на кнопку “Заказать”?

3. ***Проверенная тобой функциональность готова к релизу? Почему?***

   Функциональность не готова к релизу — есть 2 критических бага, которые заграивают основной пользовательский сценарий. Один из них делает функциональность нерабочей. Также есть много багов со средним приоритетом.

Ретест багов в мобильном приложении

Был проверен фикс багов. Из них не исправлено **1**, исправлено — **3**, найден **1**.

Список багов можно посмотреть здесь:

- Исправлено
- Исправлено
- Исправлено
- Не исправлено
- Новый баг

Регрессионное тестирование мобильного приложения по готовым тест-кейсам

Результаты выполнения регрессионных тестов можно посмотреть здесь:
Гончаров Сергей, 13-я когорта - регрессионное тестирование Самокат

Из **10** успешно прошло **1**, не прошло — **9**.

Список багов, найденных при тестировании, разбит по приоритетам:

1. Блокирующие:

- Баг 1

2. Критичные:

- Баг 1
- Баг 2
- Баг 3
- Баг 4

3. Средний приоритет:

- Баг 1
- Баг 2
- Баг 3

4. Низкий приоритет:

- Баг 1

Заключение:

1. ***Какой баг показался самым критичным?***

   Все выявленные критические баги так или иначе мешают пользователю использовать главный функционал приложения. Однако ошибки связанные с отображением информации о заказе делают использование продукта невозможным, ибо курьер не сможет выполнить заказ.

2. ***Такой продукт можно выпускать в релиз? Почему?***

   Такой продукт выпускать в релиз не стоит, ибо многие выявленные ошибки затрагивают основной пользовательский сценарий, а некоторые делают использование этого продукта и вовсе невозможным.

Выводы о проделанной работе

Как для тебя прошла первая практическая часть проекта? С какими сложностями пришлось столкнуться? Что получилось хорошо, а что не очень? Какие мысли остались?

Это были интересные задания, представляющие собой некий срез всего материала за курс.Эта часть не показалась мне сложной. После разделения задач на более мелкие и составления для себя плана по их выполнению, всё стало вполне понятно и очевидно.

Что получилось хорошо: Старался делать как можно более наглядную и информативную документацию и отчетность - делить на блоки, выделять цветами, прикладывать криншоты, скринкасты и логи. Также получилось выполнить задания немного быстрее, чем я планировал изначально.

Сложности: В процессе выявил пробелы в знаниях тестирования мобильных приложений, ибо впервые решил проводить тесты на реальном устройстве. Думаю, новые знания и опыт пойдут мне на пользу.