**Для POST /api/customer/{customer\_id}/purchase**

Проверка на корректность начисления баллов:

Проверяем, что при покупке на 500 рублей баллы не начисляются, при покупке на 1000 рублей должен начисляться 1 балл, а при покупке на 2500 рублей - 2 балла.

Так же нужно проверить начисление баллов более крупных сум: например, при покупке на 100000 рублей.

Проверка на обработку граничных значений:

Проверка начисления баллов: при покупке на 999 рублей не должно начисляться и на 1000 рублей должен начисляться 1 балл.

Проверка на отсутствие округления: покупка на 1999 рублей должна принести только 1 балл, а не 2.

Проверка на актуализацию данных:

После каждой покупки проверяем, что обновленный баланс баллов отображается корректно в системе, вызвав метод GET/api/customer/{customer\_id}/points после каждой транзакции.

Проверяем, что данные обновляются в реальном времени без задержек.

Проверка на устойчивость к ошибкам:

Провести тесты с невалидными данными (например, отрицательные суммы, несуществующие ID клиентов) и убедиться, что система возвращает ошибки.

Проверка обработки экстремальных значений (очень большие суммы покупок) и, что система не допускает переполнения или других проблем с обработкой данных.

**Для GET /api/customer/{customer\_id}/points**

Точность данных:

Убедимся, что после ряда покупок количество баллов, отображаемое этим методом, соответствует ожидаемому. Это требует серии покупок и проверки баланса после каждой покупки.

Проверьте, что при просмотре баллов разными клиентами данные не перемешиваются или не отображаются некорректно.

Кеширование:

Если в системе есть кэширование, нужно убедиться, что после изменения баланса баллов кэш обновляется и предоставляет актуальную информацию.

Мы можем изменять баллы и проверять, что изменения отражаются быстро и корректно.

Защита доступа:

Проверяем, что пользователь не может получить доступ к баллам других клиентов, используя разные customer\_id.

Протестируем систему на предмет уязвимостей, которые могли бы позволить получить доступ к чужим баллам, например, через инъекции SQL или другие виды атак.

**Для POST /api/discount/calculate**

1. 1% (0–149 баллов):
   * Установим баланс в 0 баллов, провереям, что скидка составляет 1%.
   * Установим баланс в 149 баллов, проверьте, что скидка все еще 1%.
2. 5% (150 баллов):
   * Установим баланс в 150 баллов, проверям, что скидка увеличилась до 5%.
3. 5% (151–199 баллов):
   * Установим баланс в 151 и проверяем, что скидка 5%.
   * Повторяем для 199 баллов, проверяем, что скидка не изменилась.
4. Переход к скидке 7% (200 баллов):
   * Установим баланс в 200 баллов, проверям, что скидка теперь 7%.
5. Проверка скидки 7% (201–399 баллов):
   * Установим баланс в 201 и 399 баллов, проверяем, что скидка остается 7%.
6. Переход к скидке 10% (400 баллов):
   * Установите баланс в 400 баллов, проверяем, что скидка увеличилась до 10%.

**Общие проверки**

1. **Интеграционное тестирование:**

Проверяем, что все методы API работают корректно вместе. Например, после того как баллы начислены с помощью POST /api/customer/{customer\_id}/purchase, они должны корректно отображаться в GET /api/customer/{customer\_id}/points и корректно влиять на расчет скидок в POST /api/discount/calculate.

Тестируем сценарии, в которых последовательно выполняются операции покупки, проверки баллов и расчета скидок.

1. **Нагрузочное тестирование:**

Проверяем, как система справляется с большим количеством одновременных запросов. Это покажет, насколько хорошо система масштабируется и готова к реальной эксплуатации.