**Вводные данные:**

Перед Вами поставлена задача реализовать бэк для страницы оплаты просмотра фильма через некий платёжный сервис **А**. Рассмотрим упрощённую схему, подразумевающую, что сервис сразу отвечает нам о возможности списания средств, без дополнительной авторизации через 3D-Secure.

Фронт для этой страницы уже реализован, он передаёт JSON-объект, содержащий текстовые поля *card\_number* (номер карты, 16 цифр, считаем, что валидный номер всегда содержит 16 цифр, валидность номера по алгоритму Луна проверять не нужно), *card\_holder* (владелец карты, имя и фамилия латиницей, может также содержать дефис), *card\_expiration* (месяц/год окончания действия карты в формате мм/гг), *cvv* (код с обратной стороны карты, 3 цифры), *order\_number*(номер заказа, до 16 произвольных символов), *sum* (сумма оплаты, разделитель дробной и целой части **запятая**, поэтому и строка, а не число).

Бэк состоит из одного метода контроллера и выполняет следующие действия:

* Валидирует данные, если в данных есть ошибка, то возвращает сообщение об ошибке с кодом 400;
* Если данные верные, то передаёт их в API-запросе на сервис **A**. Сервис **A** пытается списать деньги, если ему это не удаётся, то он возвращает HTTP-код 403, если удаётся, то HTTP-код 200;
* В случае ошибки передаём её обратно на фронт.
* В случае успешного списания денег необходимо записать в БД информацию об успешной оплате. Предполагаем, что у нас есть соответствующий метод репозитория setOrderIsPaid(string $orderNumber, float $sum): bool, реализованный ранее. Метод проверяет соответствие номера заказа и его суммы и возвращает true, если списание успешно. В случае ошибок выбрасываются различные исключения.

**Задача:**

Предлагается описать кейсы тестирования данной задачи для трёх уровней: модульного, интеграционного и системного. Считаем, что взаимодействуют 4 «модуля»: бэк, фронт, репозиторий и сервис **A**.

Кейсы тестирования:

1. Модульные:

а) Если *card\_number* не содержит ровно 16 цифр и только их (проверяем регулярным выражением), то тестируемый метод возвращает HTTP-код 400 с сообщением об ошибке;

б) Если *card\_holder* содержит более одного пробела, то тестируемый метод возвращает HTTP-код 400 с сообщением об ошибке;

в) Если *card\_holder* содержит более 1 дефиса, то тестируемый метод возвращает HTTP-код 400 с сообщением об ошибке;

г) Если *card\_holder* содержит символ не входящий в разрешенный список (латиница), то тестируемый метод возвращает HTTP-код 400 с сообщением об ошибке;

д) Если *card\_expiration* не соответсвует формату мм/гг (проверяем регулярным выражением 2 цифры 01 до 12, 2 цифры), то тестируемый метод возвращает HTTP-код 400 с сообщением об ошибке;

е) Если *card\_expiration* соответсвует формату мм/гг (проверяем регулярным выражением 2 цифры 01 до 12, 2 цифры), но дата старее текущей даты (например сегодня 09/19, а введено 08/19), то тестируемый метод возвращает HTTP-код 400 с сообщением об ошибке;

ж) Если *cvv* не равен 3м цифрам, то тестируемый метод возвращает HTTP-код 400 с сообщением об ошибке;

з) Если длина *order\_number* более 16, то тестируемый метод возвращает HTTP-код 400 с сообщением об ошибке;

и) Если *order\_number* содержит недопустимые символы, то тестируемый метод возвращает HTTP-код 400 с сообщением об ошибке;

к) Если *sum* это не строка, то тестируемый метод возвращает HTTP-код 400 с сообщением об ошибке;

л) Если *sum* содержит больше одной запятой (разделитель дробной части), то тестируемый метод возвращает HTTP-код 400 с сообщением об ошибке;

м) Если *sum* содержит символы не цифры и не запятую, то тестируемый метод возвращает HTTP-код 400 с сообщением об ошибке;

2) Интеграционные:

Модуль – сервис А

а) модуль передает данные на сервис А. Если во время передачи данных произошла ошибка (разрыв коннекта, не коннект и тп), то после получения ответа от бэка показываем на фронте информационное сообщение пользователю;

б) модель успешно передал данные на сервис А и сервис вернул нам ответ, что он не списал деньги, то тестируемый метод возвращает HTTP-код 403 с сообщением об ошибке и после получения ответа от бэка показываем на фронте информационное сообщение пользователю;

в) модуль успешно передал данные на сервис А и сервис вернул нам ответ, что он успешно списал деньги, то тестируемый метод возвращает HTTP-код 200 с сообщением об успешном списании денег и после получения ответа от бэка показываем на фронте информационное сообщение пользователю;

г) модель успешно передал данные на сервис А и сервис вернул нам ответ, но это не сообщение о невозможности списать денег или о том, что он успешно их списал, то тестируемый метод возвращает HTTP-код 400 с сообщением об ошибке и показываем на фронте информационное сообщение пользователю;

Модуль – Репозиторий:

а) наш модуль передает на setOrderIsPaid валидные номер ордера (строка) и (float) сумму для проверки корректности суммы и номера ордера (что эта сумма для этого ордера). Если ошибка, то возвращаем код ошибки и сообщение об ошибке, после получения ответа от бэка показываем на фронте информационное сообщение пользователю.

б) наш модуль передает на setOrderIsPaid валидные номер ордера (строка) и (float) сумму для проверки корректности суммы и номера ордера (что эта сумма для этого ордера) и получили ответ true, передаем его на фронт и после получения ответа от бэка, переадресуем пользователя на страницу информирующую об успешной оплате.

Фронт – Бэк:

а) Если *card\_number* не содержит ровно 16 цифр и только их, то после получения ответа от бэка выделяем поле с *card\_number*  красной рамкой;

б) Если *card\_holder* содержит более одного пробела, то после получения ответа от бэка выделяем поле с *card\_holder* красной рамкой;

в) Если *card\_holder* содержит более 1 дефиса, то после получения ответа от бэка выделяем поле с *card\_holder* красной рамкой;

г) Если *card\_holder* содержит символ не входящий в разрешенный список, то после получения ответа от бэка выделяем поле с *card\_holder* красной рамкой;

д) Если *card\_expiration* не соответсвует формату мм/гг (проверяем регулярным выражением 2 цифры 01 до 12, 2 цифры, то после получения ответа от бэка выделяем поле с *card\_expiration* красной рамкой;

е) Если *card\_expiration* соответсвует формату мм/гг (проверяем регулярным выражением 2 цифры 01 до 12, 2 цифры), но дата старее текущей даты (например сегодня 09/19, а введено 08/19), то после получения ответа от бэка выделяем поле с *card\_expiration* красной;

ж) Если *cvv* не равен 3м цифрам, то после получения ответа от бэка выделяем поле с *cvv* красной рамкой;

з) Если длина *order\_number* более 16, то после получения ответа от бэка выделяем поле с *order\_number*  красной рамкой;

и) Если *order\_number* содержит недопустимые символы, то после получения ответа от бэка выделяем поле с *order\_number*  красной рамкой;

к) Если бэк вернул ошибку по полю *sum*, то после получения ответа от бэка выделяем поле *sum*  красной рамкой;

л) Если *sum* содержит больше одной запятой (разделитель дробной части), то после получения ответа от бэка выделяем поле *sum*  красной рамкой;

м) Если *sum* содержит символы не цифры и не запятую, то после получения ответа от бэка выделяем поле *sum*  красной рамкой;

н) Если *sum* не соответвсвует номеру ордера, то после получения ответа от бэка выделяем поле *sum*  красной рамкой;

1. Системные:

а) если во время передачи данных на сервис А произошла ошибка, то показать сообщение «К сожалению, мы не можем обработать платеж в данный момент, пожалуйста, попробуйте немного позже»;

б) оплата прошла успешно, вывести сообщение об успешной оплате: «Оплата прошла успешно. Приятного просмотра и ждем Вас снова»;

в) если сумма ордера и номер не соответствуют, то вывести сообщение: «Сумма ордера не верна. Создайте ордер заново или перепроверьте сумму.»

г) Если *card\_number* не содержит ровно 16 цифр то пишем сообщение об ошибке: «Номер карты должен быть только 16 цифр»;

д) Если *card\_holder* содержит более одного пробела, пишем сообщение об ошибке: «Поле card holder может содержать только 1 пробел»;

е) Если *card\_holder* содержит более 1 дефиса то пишем сообщение об ошибке: «Поле card holder может содержать не более одного дефиса»;

ж) Если *card\_holder* содержит символ не входящий в разрешенный список, то выделяем поле с *card\_holder* красной рамкой и пишем сообщение об ошибке, которое передал бэк;

з) Если *card\_expiration* не соответсвует формату мм/гг пишем сообщение об ошибке: «Не верный формат даты»

и) Если *card\_expiration* соответсвует формату мм/гг (проверяем регулярным выражением 2 цифры 01 до 12, 2 цифры), но дата старее текущей даты (например сегодня 09/19, а введено 08/19), то пишем сообщение об ошибке: «Вы не можете указать даты из прошлого»;

к) Если *cvv* не равен 3м цифрам, то пишем сообщение: «Cvv должно быть только 3 цифры»;

л) Если длина *order\_number* не равно16, то пишем сообщение об ошибке: «Не верный номер ордера»;

м) Если *order\_number* содержит недопустимые символы, то пишем сообщение об ошибке: «Не верный номер ордера»;

н) Если *sum* содержит больше одной запятой (разделитель дробной части), то пишем сообщение об ошибке: «Формат суммы не верный»;

о) Если *sum* содержит символы не цифры и не запятую, то пишем сообщение об ошибке: «Формат суммы не верный»;

п) Если *sum* не соответствует номеру ордера, пишем сообщение об ошибке: «Несоответствие суммы и ордера»;

Этот вопрос не понятен:

1. Поскольку мы тестируем оплату, то стоимость тестирования в некоторых ситуациях будет выражаться в реальных денежных затратах, а не в виртуальных. Стоит обратить на это внимание и, возможно, предложить способы удешевления тестирования.

Во всех платежных системах есть тестовые варианты оплаты. Так же есть валидные номера карты виза типа 4222222222222 и не валидные. Если уж очень хочется платить реальными деньгами, то ставим оплату минимально возможную в данной платежной системе. Если это можно, то 1 копейку. На рубль можно попробовать 100 раз.