Мысли об автоматическом тестировании сервиса по регистрации мишек

Документ является верхнеуровневым описанием подходов, которые я применил бы для автоматического тестировании сервиса по регистрации мишек.

Прилагающийся код является прототипом, который, с одной стороны, позволяет реализовать все описанные в документе подходы, с другой стороны, не является готовым решением для тестирования: в нём не настроено логирование, недостаточно гибкие фикстуры, и. т. д.

0.
Пара определений в самом начале.

Корректный мишка – это мишка с корректным именем, корректным типом и корректным возрастом.

Корректное имя
- состоит из допустимого набора символов
- имеет длину, не превышающую некоторого максимального значения.

Корректный тип задаётся перечислением.

Корректный возраст
- задаётся вещественным числом (в формате IEEE 754) с фиксированной максимальной точностью,
- не меньше нуля,
- не превышает некоторого максимального значения, будем считать, что не зависящего от типа мишки.

Случайный мишка – мишка с рандомно сгенерированными именем, типом и возрастом. При необходимости для создания новых мишек можно использовать кастомные генераторы, с фиксированным сидом.

1.

Для тестирования запросов

- post /bear,
- put /bear/:id,

получающих значение параметров bear_name, bear_type, bear_age из пейлоада запроса.

1.1. Проверяем работу запросов с корректными мишками.

Пеирвайзим тестовый набор корректных мишек на основе следующих данных:

варианты имени мишки:

- пустая строка
- корректное имя средней длины
- корректное имя максимальной длины

варианты типа мишки:

- возможны все допустимые типы

варианты возраста мишки:

- нулевой возраст
- возраст близкий (в смысле IEEE 754) к нулевому
- средний возраст с нулевой дробной частью
- средний возраст с дробной частью максимальной точности
- максимально возможный возраст
- возраст близкий (в смысле IEEE 754) к максимально допустимому возрасту медведя

1.1.1. Для тестирования запроса post /bear

Отдельным тестом постим каждого мишку из сгенерированного набора в отдельном экземпляре сервиса.

Убеждаемся в том, что

- мишка появился в системе,
- сервис присвоил мишке id = 1
- сервис корректно возвращает имя, тип и возраст сохраненного мишки.

1.1.2. Для тестирования запроса put /bear/:id

Отдельным тестом заменяем случайного мишку на мишку из сгенерированного набора в отдельном экземпляре сервиса.

Для каждого мишки убеждаемся в том, что

- мишка появился в системе,
- сервис присвоил мишке id = 1
- сервис корректно возвращает имя, тип и возраст сохраненного мишки.

Убеждаемся в том, что
- параметры мишки изменились ,
- id мишки остался неизменным,
- сервис корректно возвращает имя, тип и возраст измененного мишки.

1.2. Проверяем работу запросов с None параметрами.

Быстрый вариант:

1.2.1. Для тестирования запроса post /bear

Пытаемся создать трёх неправильных мишек.
- случайного мишку с именем None, с корректным возрастом и корректным типом,
- случайного мишку с возрастом None, с корректным именем и корректным типом,
- случайного мишку с типом None, с корректным возрастом и корректным возрастом.

Проверяем коды возвращаемых ошибок и текст респонсов.

1.2.2. Для тестирования запроса put /bear/:id

Создаём случайного корректного мишку. Пытаемся поменять ему
- значение имени на None,
- значение возраста на None,
- значение типа на None,

оставив все остальные параметры без изменений.

Подробный вариант:

Перебираем все подмножества трехэлементного множества параметров: {bear_name, bear_type, bear_age}, кроме пустого.

Для каждого выбранного подмножества параметров пытаемся создать некорректного мишку, у которого все параметры из выбранного подмножества имеют значение None.

1.2.3. Для тестирования запроса post /bear

Для каждого выбранного подмножества пытаемся создать некорректного мишку, значения соответствующих параметров которого равно None. Проверяем код возвращаемых[ошибок и текст респонсов.

1.2.4. Для тестирования запроса put /bear/:id

Создаём случайного корректного мишку. Для каждого выбранного подмножества пытаемся пытаемся заменить соответствующие параметры случайного мишки на None. Проверяем коды возвращаемых ошибок и текст респонсов.

1.3. Проверяем работу запросов с отсутствующими параметрами.

Аналогично предыдущему пункту, только вместо того, чтобы присваивать параметру значение None, исключаем этот параметр из пейлоада соответствующего запроса.

1.4. Проверяем работу запросов с дублирующимися именами мишек.

С помощью запросов

- post /bear и
- put /bear/:id

пробуем создать двух мишек

- с одинаковым именем и с разными значениями типа и возраста,
- с одинаковым именем и с одинаковыми значениями типа и возраста (мишки дубликаты),
- с одинаковым именем и с одинаковыми значениями типа, но с разными значениями возраста,
- с одинаковым именем и с одинаковыми значениями возраста, но с разными значениями типа.

1.5. Проверяем работу запросов с некорректно заданными именами мишек.

Необходимо проверить имена

- с недопустимыми символами,
- с различными разделительными символами в середине имени,

- с различными разделительными символами в начале или конце имени (хотелось бы, чтобы тримминг работал),
- слишком длинные (с точки зрения модели данных).

1.6. Проверяем работу запросов с некорректно заданными типами мишек.

- Необходимо проверить любой тип, не относящийся к списку корректных типов.

1.7. Проверяем работу запросов с некорректно заданными возрастами мишек

Необходимо проверить возрасты

- заданные с помощью int-ов, которые неявно преобразуются к float,
    - с корректным значением возраста после преобразования,
    - с некорректным значением возраста после преобразования,
- заданные с помощью строк, содержащих различные символьные представления int или float (список возможных представлений может быть довольно большим),
    - с корректным значением возраста после преобразования,
    - с некорректным значением возраста после преобразования,
- отрицательный возраст,
- возраст, превышающий максимально допустимый (с точки зрения модели данных) возраст медведя,
- возраст, по модулю выходящий за пределы Float.MAX_VALUE в Java (насколько я понял, там джарник внутри контейнера),
- возраст, заданный с точностью, превышающей максимально допустимую (с точки зрения модели данных).

2.
Для тестирования запросов

- get /bear/:id,
- put /bear/:id,
- delete /bear/:id,

получающих значение параметра id из url, проводим следующие проверки.

2.1.
Выполнить запрос со значением параметра id, соответствующим идентификатору одного из зарегистрированных в системе мишек.

2.2.
Выполнить запрос со значением параметра id, не соответствующим идентификатору какого-либо зарегистрированного в системе мишки.

2.3.
Выполнить запрос со значением параметра

- id = 0,
- id > 0

при отсутствии зарегистрированных в системе мишек.

2.4.
Выполнить запрос со значением параметра id, лежащего за границей диапазона значений, допустимых для соответствующего типа переменной, объявленной в хендлере запроса (например, Integer.MAX_VALUE).

2.5.
Для тестирования запроса put /:id дополнительно проверить передачу пары несогласованных между собой значений параметра

- id в url запроса и
- bear_id в пейлоаде запроса (в качестве избыточного параметра).

    2.5.1.:
    - валидное значение параметра id,
    - валидное значение параметра bear_id.

    2.5.2.:
    - валидное значение параметра id,
    - невалидное значение параметра bear_id.

    2.5.3.:
    - невалидное значение параметра id,
    - валидное значение параметра bear_id.

    2.5.4.:
    - невалидное значение параметра id,
    - невалидное значение параметра bear_id.

2.6.
В случае, если мишки разных типов по-разному обрабатываются в системе, необходимо проверить эти запросы, передав им id, соответствующие мишкам каждого типа.

3.
Кроме того, необходимо убедиться, что корректно игнорируется лишний параметр bear_id, переданный в пейлоаде запроса post /bear.

4.
Для тестирования get запросов

- get /bear/:id
- get /bear

проверяем возможное кэширование результатов выполнения get запросов на стороне бэкэнда.

4.1. Для тестирования запроса get /bear/:id.

Стартуем две клиентских сессии. Из первый клиентской сессии создаём, модифицируем и удаляем одного и того же мишку.

Из обеих сессий запускаем запрос get /bear/:id с идентификатором этого мишки, проверяем, что каждый запрос возвращает о мишке актуальные данные.

4.1.1.
После создания мишки из первой клиентской сессии, мишка начинает отображаться в результатах запросов get /bear/:id, запущенных из любой из двух клиентских сессий.

4.1.2.
После модификации мишки из первой клиентской сессии, мишка обновляется в результатах запросов get /bear/:id, запущенных из любой из двух клиентских сессий.

4.1.3
После удаления мишки из первой клиентской сессии, мишка перестаёт отображаться в результатах запросов get /bear/:id, запущенных из любой из двух клиентских сессий.

4.2. Для тестирования запроса get /bear

Стартуем две клиентских сессии. Из первой клиентской сессии создаём, модифицируем и удаляем нескольких мишек.

Из обеих сессий запускаем запрос get /bear, проверяем, что каждый запрос возвращает актуальные данные о каждом из мишек.

4.2.1.
После создания нового мишки из первой клиентской сессии, мишка начинает отображаться в результатах запросов get /bear, запущенных из любой из двух клиентских сессий.

4.2.2.
После модификации мишки из первой клиентской сессии, мишка обновляется в результатах запросов get /bear, запущенных из любой из двух клиентских сессий.

4.2.3
После удаления мишки из первой клиентской сессии, мишка перестаёт отображаться в результатах запросов get /bear, запущенных из любой из двух клиентских сессий.

5.
Для тестирования запросов

- get /bear
- delete /bear

необходимо проверить на корректность работы с очень большим количеством мишек, одновременно зарегистрированных в системе.

6.
Для тестирования запроса get /info необходимо убедиться, что этот запрос выполняется и возвращает правильный html.