**Alaska tests**

**Bugs and recommendations**

1. **Instruction's spelling mistakes**

Steps:

GET [<http://127.0.0.1:8091/info>](<http://127.0.0.1:8091/info>)

Actual result:

Изображение выглядит как текст

Автоматически созданное описание

Expected result:

information without literal errors

Note. Make sure we need colon before ids in documentation (for example, PUT /bear/:id)

Priority: low

**2. Make the error message more specific and formal, change the status**

Steps:

POST <http://127.0.0.1:8091/bear> (empty body)

Actual result:

Response: Error. Pls fill all parameters

Status: 200 OK

Expected result:

formal message

Note.

POST <http://127.0.0.1:8091/bear> {"bear\_type":"GUMMY","bear\_name":"mikhail"} (without bear\_age)

→ Error. Pls fill all parameters

It is not clear which parameter is missed

Priority: medium

**3. It could be better if successful PUT and DELETE requests would return 204 status instead of 200. The response body contains only 'OK' and does not have any additional information. 204 No Content could be a better idea.**

Precondition:

there is a bear with id = **existing\_bear\_id**

Steps:

PUT <http://127.0.0.1:8091/bear/>**existing\_bear\_id** {"bear\_type":"null","bear\_name":"mikhail","bear\_age":17.5}

Actual result:

200 OK, Response: OK

Expected result:

204 No cont

Priority: low

**4. User sees HTML tags from the response in case of errors no matter which format was chosen. Also, there are only two types of error codes for any invalid input, that is not enough. We need more specific errors and descriptions of what went wrong.**

Steps:

POST <http://127.0.0.1:8091/bear>, {"bear\_type":"PANDA","bear\_name":"mikhail","bear\_age":17.5} *(bear\_type is not from list)*

Actual result:

Response: <html><body><h2>500 Internal Server Error</h2></body></html>

Expected result:

specific errors and status codes, no tags

Note. Another example of error message where it's not clear what went wrong (bear\_id has incorrect value - '/').

PUT [http://127.0.0.1:8091/bear/\\](http://127.0.0.1:8091/bear/%5C%5C) {"bear\_type":"null","bear\_name":"mikhail","bear\_age":17.5}

→ <html><body><h2>404 Not found</h2></body></html>

Priority: low

**6. It could be a better idea if bear\_age field accepted only integer values**

Steps:

POST <http://127.0.0.1:8091/bear> {"bear\_type":"BLACK","bear\_name":"mikhail","bear\_age":14.4}

Actual result:

Status: 200 OK

Expected result:

impossible to create a bear with not integer age

Priority: low

**7. Set maximum value for the bear\_age field**

Steps:

POST <http://127.0.0.1:8091/bear> {"bear\_type":"BLACK","bear\_name":"Anna","bear\_age":400}

Actual result:

Status: 200 OK

Expected result:

impossible to create a bear with age greater than some value that would be specified

Priority: medium

**8. It could be better if bear\_name field accepted only alphabetical values**

Steps:

POST <http://127.0.0.1:8091/bear>

{"bear\_type":"BLACK","bear\_name":11,"bear\_age":15}

Actual result:

Status: 200 OK

Expected result:

impossible to create a bear with integer name

Priority: medium

**9. Set a minimal amount of characters for a bear\_name filed. Set maximum amount of characters as well.**

Steps:

POST <http://127.0.0.1:8091/bear>

{"bear\_type":"BLACK","bear\_name":"","bear\_age":null}

Actual result:

Status: 200 OK

Expected result:

impossible to create a bear with the name which amount of characters is invalid

Priority: medium

**10. Bear's name in uppercase while we created (POST) it lowercased.**

Note. PUT allows lowercase. It's better not change the name's case

Steps:

POST <http://127.0.0.1:8091/bear> {"bear\_type":"BLACK","bear\_name":"mikhail","bear\_age":17.5}

→ **bear\_id**

GET <http://127.0.0.1:8091/bear/>**bear\_id**

Actual result:

Response: {"bear\_id":1,"bear\_type":"BLACK","bear\_name":"MIKHAIL","bear\_age":17.5}

Expected result:

"bear\_name":"mikhail"

Priority: medium

**11. Null response appears when a user creates and reads a bear with bear\_type = "GUMMY". This could be a bug in requirements. Should be a gummy bear in Alaska? Let's imagine it is our feature to entertain children, for example. Anyway, the system should not create an entry and return null after reading.**

Steps:

POST <http://127.0.0.1:8091/bear> {"bear\_type":"GUMMY","bear\_name":"mikhail","bear\_age":17.5}

→ **bear\_id**

GET <http://127.0.0.1:8091/bear/>**bear\_id**

Actual result:

Response: null

Expected result:

{"bear\_id":**bear\_id**, "bear\_type":"GUMMY","bear\_name":"mikhail","bear\_age":17.5}

Priority: high

**12. It could be better to let a user know that the table was already empty, and change response and status**

1. Steps:

DELETE <http://127.0.0.1:8091/bear/> - while the table is already empty

Actual result:

Response: OK, Status: 200 OK

Expected result:

specific message

1. Steps: send

DELETE <http://127.0.0.1:8091/bear/>**not\_existing\_id**

Actual result:

Response: OK, Status: 200 OK

Expected result:

specific message

Priority: low

**13. bear\_type and bear\_name could not be changed**

Steps:

POST <http://127.0.0.1:8091/bear> {"bear\_type":"POLAR","bear\_name":"mikhail","bear\_age":17.5}

→ **bear\_id**

PUT <http://127.0.0.1:8091/bear>/**bear\_id**

{"bear\_type":"BROWN","bear\_name":"Masha","bear\_age":10.5}

GET <http://127.0.0.1:8091/bear/>**bear\_id**

Actual result:

Response {"bear\_id":57,"bear\_type":"POLAR","bear\_name":"Masha","bear\_age":17.5}

Priority: high

**14. 'bear\_' is redundant in parameters id, type, name, age. It could be better not to start with this substring. We have only bears but if we had more species 'bear\_' would more valid. I suppose that Alaska service might have a wider functionality, and we can need 'bear\_'.**

Priority: low

**15. It could be a better idea to make the response more specific, it's not clear what a number from the response means.**

Steps:

POST <http://127.0.0.1:8091/bear>

{"bear\_type":"BLACK","bear\_name":"mikhail","bear\_age":17.5}

Actual result:

Response: 1

Expected result: "id": 1 (or "bear\_id": 1)

Priority: low

**16. Security problems. There is no authorization. Anyone can delete all the entries, for example. It's better to provide security. Also, we use the HTTP protocol, which is not quite safe.**

Priority: high

**17. Text/html format returns by default. I get "text/html;charset=utf-8" format when I run the tests. I would prefer JSON format by default. But maybe we do need HTML. I would discuss this.**

Priority: low

**18. All bear\_types are in Uppercase. I would discuss if we need this implementation.**

Priority: low

**19. It would be nice to have a possibility of updating bear's fields not providing all the fields via the PUT method.**

Priority: low

**Checklist**

**GET /info**

status code, response body (in different formats)

**GET /bear**

empty database, 1 item, several items, a lot of items (pagination?), read after invalid creation,

status code, response body (in different formats)

**GET /bear/:id**

existing item, not existing item, created and deleted item.

XSS injection: "<script>alert(\"code!\")</script>",

status code, response body (in different formats)

**Examples of parameter:**

**id:** 1, 0, -1, null, 1.5, a, 17,5; 999..., "14.13", 1.0E-999..., 1.0E999..., "a", special characters, spaces, object {}, array [], /n

**POST /bear**

valid input, invalid input, boundary values, optional parameters, id incrementation, object {}, array[], special characters, spaces, additional parameters, XSS injection

empty body,

add path parameter,

change parameters order,

body without bear\_age, body without bear\_name, body without bear\_type,

check status codes,

send a request 2 times, update one bear twice with different parameters,

delete all records and send a request, delete 1 record and send the request (+ check ids),

XSS injection: "<script>alert(\"code!\")</script>" as a parameter,

create 1000, 100000 items, check the time of all the operations on a large number of values.

**Examples of parameters:**

**"bear\_type"**: "POLAR", "BROWN", "BLACK", "GUMMY", "", "PANDA", " ", " BLACK", "BROWN ", 'polar', long string, "a"

Note. When it's only four types we can easily cover them, but if there would be a lot of types, we would not check all of them. Equivalence class partitioning would help us.

**"bear\_name"**: "mikhail", "Anna", 'MARK", 11, "", " ", "m", " h", "h ", null, long string, /n, special characters, null, object {}, array [], another alphabet values - "Маша"

**"bear\_age"**: 17.5; 17,5; "14.13", 0, 0.1, -0.1, 1, -1, null, 999.., 1.0E-999..., 1.0E999..., "a", object {}, array [], /n

**PUT /bear/:id**

send a request with existing id,

send a request with not existing id,

send a request with not existing id that was previously added and deleted,

send a request twice (idempotency),

send a request without body,

send a request without bear\_type, send a request without bear\_name, send a request without bear\_age,

send a request with "bear\_id" as a query parameter,

change parameters order,

check responses (in different formats) and statuses.

**Examples of parameter:**

**id** - see GET request's examples

**"bear\_type"**: valid and invalid values (see examples from POST /bear), change the value from one to another and return to the first, update GUMMY bear (reading returns a null value), update not GUMMY bear to GUMMY bear,

**"bear\_name"**: valid and invalid values (see examples from POST /bear), change the value from one to another and return to the first,

**"bear\_age"**: valid and invalid values (see examples from POST /bear), change the value from one to another and return to the first.

**DELETE /bear**

Send the request while the list is not empty, send a request while the list is empty, send the request twice, check response (in different formats) and status, delete GUMMY bear (reading returns a null value)

**DELETE /bear/:id**

send a request with existing id,

send a request with not existing id,

send a request twice,

send a request with not existing id that was previously added and deleted,

check responses (in different formats) and statuses,

delete GUMMY bear (reading returns a null value)

**Examples of parameter:**

**id** - see GET request's examples

**Code example**

There are several examples of autotests that could be implemented.

Note. Tests bearWithValidParametersStoresCorrectlyAfterCreation, bearCanBeUpdated, listOfbearsWithValidParametersStoresCorrectlyAfterCreation fail because of bugs in the service. Also, I do not like how this service provides, so I decided not to add autotests covering incorrect input.

