RESULT PRESENTATION OF MAIN TASK

According to the MAIN TASK following functionality were tested:

- Create new pet and verify whether it was created or not;
- Update the name of the pet and verify whether it was changed or not;
- Delete the pet and verify whether it was deleted or not.

During requirements investigation and exploratory testing of server part of the application I separated additional flows for better test coverage. So I decided to concentrate into the following areas of application:

- New pet creation;
- Updating of pet;
- Deleting of pet;
- Searching for pets.

New pet creation

Let's start with new pet creation functionality. Pet creation functionality was divided into the following smaller areas:

- Positive testing cases;
- Negative testing cases;
- Validation;
- Tags sorting.

Positive testing cases: I verified the cases when the pet is created with the several tags and the several URLs because these fields can be saved in the DB as array of data. So in this section you can find the requests for creating (post request) and verifying these flows (get request). Any defects weren't found in these flows.

Negative testing cases: In this section I verified the cases when the pet is created with added the same tags, URLs or with non-existing status in the system. According to the requirements "Status" field has string format, but only 3 values are valid for this field – available, pending and sold. During testing of this part of the functionality the defect were found and reported into the Defect Reports document.

Validation: In order to receive the better test coverage I decided to divide this area too into smaller parts:

- Form fields validation (all verifications related to creating pet with or without required fields);
- Fields validation (all verifications related to format of the data that can be put into the fields)

I used the decision table technic in order to create test for form fields validation. So my decision table for this functionality was following:

	1	**2**	**3**	**4**
Required fields are filled	Y	Y	N	N

	1	2	3	4
Non-required fields are filled	Y	N	Y	N
Output:	1) Pet will be created. 2) API response with “200” code will be shown	1) Pet will be created. 2) API response with “200” code will be shown	1) Pet won’t be created; 2) API response with “404” code will be shown	1) Pet won’t be created; 2) API response with “404” code will be shown

According to the table below case_1 and case_2 are related to positive testing cases and case_3 and case_4 are related to negative testing cases. Besides that the set of negative tests were increased by the verification for quantity of filled required fields. According to requirements “Pet” model has 2 required fields – “Name” and “PhotoUrls” fields. So using the same technic (decision table) following tests were created too:

	1	2	3	4
‘Name’ field	Y	Y	N	N
‘PhotoUrls’ field	Y	N	Y	N
Output:	1) Pet will be created. 2) API response with “200” code will be shown	1) Pet won’t be created; 2) API response with “404” code will be shown	1) Pet won’t be created; 2) API response with “404” code will be shown	1) Pet won’t be created; 2) API response with “404” code will be shown

So case_1 and case_4 were covered by previous tests. And negative testing cases were expanded by case_2 and case_3.

For testing the fields validation I used the boundary values (BV) equivalence partitioning (EP) technics. For better test coverage I divided the tests by the type of the fields – for numeric fields (integer type) and for text fields (string type).

Verification for numeric fields was separated into type of entered data and format of entered data.

EP for numeric fields were following by type of entered data:

		Output:
Valid classes	Numbers (0,1,2,3,4,5,6,7,8,9)	1) Pet will be created. 2) API response with “200” code will be shown
Invalid classes	Text, symbols, decimal numbers	1) Pet won’t be created; 2) API response with “404” code will be shown

BV for numeric fields were following by type of entered data:

BV	Test data	Type of classes	Output
Max value	9223372036854775807	Valid	1) Pet will be created. 2) API response with “200” code will be shown
Max-1 value	9223372036854775806	Valid	1) Pet will be created. 2) API response with “200” code will be shown
Max+1 value	9223372036854775808	Invalid	1) Pet won’t be created; 2) API response with “404” code will be shown
Min value	-9223372036854775808	Valid	1) Pet will be created. 2) API response with “200” code will be shown
Min+1 value	-9223372036854775807	Valid	1) Pet will be created. 2) API response with “200” code will be shown
Min-1 value	-9223372036854775809	Invalid	1) Pet won’t be created; 2) API response with “404” code will be shown
Max length of field	-9223372036854775808	Valid	1) Pet will be created. 2) API response with “200” code will be shown
Max+1 length of field	-92233720368547758081	Invalid	1) Pet won’t be created; 2) API response with “404” code will be shown
Min length of field	1	Valid	1) Pet will be created. 2) API response with “200” code will be shown

Besides that verifications saving of “0” value was also checked.

The validation of numeric fields by format of entered data was checked with following tests:

1) saving the “0” value in the beginning, in the middle and at the end of the field for both positive and negative numbers – the goal of the test is to verify that “0” in the beginning of the numbers isn’t saved (in case of misprinting from the user’s side, e.g.0123 will be saved as 123) and on the other part of the fields “0” value is saved (in the middle of the field and in the end of the fields, e.g.102, 10);
2) due to localization specific of the entering numeric data following negative tests were checked, that data in the X,XXX,XXX/X XXX XXX/X.XXX.XXX formats aren’t saved

The validation for text fields was also separated into the several areas – by format of entered data and type of entered data. For these goals EP technic was used:

EP	Category	Type of classes	Output
Text data	Upper case	Valid	1) Pet will be created.

EP	Category	Type of classes	Output
			2) API response with “200” code will be shown
	Lower case	Valid	1) Pet will be created. 2) API response with “200” code will be shown
Numeric data	-	Valid	1) Pet will be created. 2) API response with “200” code will be shown
Symbols	-	Valid	1) Pet will be created. 2) API response with “200” code will be shown
Array	-	Invalid	1) Pet won’t be created; 2) API response with “404” code will be shown

Besides that text data was verified by computing industry standard (UNICOD and ASCII) and following languages were added to test:

- Cyrillic (without and with didactic symbols, e.g. й, і, ї);
- Latin (without and with didactic symbols);
- Asian hieroglyph;
- Arabic symbols

Some languages were divided into with and without didactic symbols in order to identify the defect on client side in the future too.

Tag sorting: In this area sorting of added tags was checked by their IDs so that some order in DB was for better collecting, gathering and manipulating data. If the amount of saved data will be huge in the future so it’s better to fix sorting on server side, but otherwise – on the client side.

Updating of pet

Updating of pet was also divided into some smaller parts as a positive and negative testing. In this area validation of fields wasn’t verified due to the possibility to use the same pop-up for creating and updating on the client side.

For positive testing cases following tests were created:

- Update all fields;

- Update only required fields;
- Update only non-required fields;
- Add a tag for existing pet without tag;
- Update category for existing pet without added category;
- Update name of existing pet;
- Update status for existing pet;
- Add one more tag for existing pet;
- Add one more URL for existing pet;
- Delete the tag from the existing pet if it had only one tag;
- Delete the one tag from the existing pet if it had several tags;
- Delete the one URL from the existing pet if it had several URLs

For negative testing cases following tests were created:

- Update non-existing pet;
- Delete all URLs from the existing pet;
- Delete the first tag if the several tags were added (in the model - [“”, “Tag_2”]);
- Delete the tag in the middle of all added tags (in the model – [“Tag_1”, “”, “Tag_3”];
- Delete the last tag from the all added tags (in the model – [“Tag_1”,””];
- Delete the first URL if the several URLs were added (in the model - [“”, “URL_2”]);
- Delete the URL in the middle of all added URL (in the model – [“URL_1”, “”, “URL_3”];
- Delete the last URL from the all added URL (in the model – [“URL_1”,””];
- Delete the name of the pet;
- Update the status into the non-existing

Tests related to deleting the part of the array (e.g.[“”, “Tag_2”]) are aimed to verify whether the memory of the DB is used correct.

Deleting of pet

This area is short and consists of the positive test – delete the existing pet – and negative test – delete non-existing user.

Searching for pets

This section is consists with test that verify the searching by one selected status and multiple selection (positive testing cases). Besides that one test is created for searching by non-existing status (negative testing case).

Security testing

For security testing SQL and XSS injections were used. SQL injections tests verify whether it is possible to see the inner structure of the DB using the adding ‘ or -1 to the IDs of the pet. XSS injections test verify whether <> symbols are converted on HTML or not.

CONCLUSION

As a result of completing work you can find following documents:

- Test Plan;
- Test Summary Report;
- Defect Reports;
- Repository with created requests for testing;
- Result Presentation Document