Team 2 2. Test cases

Deals API

Test case ID	**DA001 - Get the list of all deals**
Description	You can obtain the complete list of all currently existing deals.
Precondition	Some deals exist in the application.
Test data	-
Test steps	1. Send **GET** `{baseUrl}/obj/deal` request.
Expected result	• Successful code: 200 OK. • List of all deals is returned in response.

Test case ID	**DA002 - Get the list of all Deals - empty limit defaulting at 50**
Description	When Get the list of all Deals request has no limit set (it is empty), then it should show the default limit of 50 items.
Precondition	More than 50 deals exist in the system's database.
Test data	Limit parameter is chosen, but set to no value (left empty).
Test steps	1. Send **GET** `{baseUrl}/obj/deal?limit=` request.
Expected result	• Successful code: 200 OK. • A default list of 50 items is returned in the response.

Test case ID	**DA003 - Get the list of all Deals - max limit at 100**
Description	When Get the list of all Deals request has a limit set that is highter than the maximum limit, then it should still show the maximum limit of 100 items.
Precondition	More than 100 deals exist in the system's database.
Test data	Limit parameter is set to value of 101.
Test steps	1. Send **GET** `{baseUrl}/obj/deal?limit=101` request.
Expected result	• Successful code: 200 OK. • A default list of 100 items is returned in the response.

Test case ID	**DA004 - Get the list of all Deals - random valid limit**
Description	When Get the list of all Deals request has a random valid limit set (e.g.10), then it should show the same random valid number of items in the list (e.g.10).
Precondition	Minimum of a random valid numer of deals exist in the system's database.

Test data	Limit parameter is set to value of a random valid number, such as 10.
Test steps	1. Send **GET** `{baseUrl}/obj/deal?limit=10` request.
Expected result	• Successful code: 200 OK. • A random valid number of items, such as 10 in this case, is returned in the response.

Test case ID	**DA005 - Get the list of all Deals - random negative limit**
Description	When Get the list of all Deals request has a random negative limit set (e.g. -200), then it should not show any results but show an error message instead.
Precondition	Some deals exist in the system's database.
Test data	Limit parameter is set to negative value, such as -200.
Test steps	1. Send **GET** `{baseUrl}/obj/deal?limit=-200` request.
Expected result	• Database retrieval failure code: 400. • No list is returned in the response.

Test case ID	**DA006 - Get the list of all Deals - cursor default 0**
Description	When Get the list of all Deals request has the cursor set with no limit (left empty), then it should default to 0.
Precondition	Some deals exist in the system's database.
Test data	Cursor parameter is chosen, but set to no value (left empty).
Test steps	1. Send **GET** `{baseUrl}/obj/deal?cursor=` request.
Expected result	• Successful code: 200 OK. • In the response, a list of items is returned starting from cursor 0, which is set by default.

Test case ID	**DA007 - Get the list of all Deals - cursor more than deals exist**
Description	When Get the list of all Deals request has the cursor set to a higher value than there are items (or is a limit set), such as 500 in this case, then no results should be returned.
Precondition	Some deals exist in the system's database.
Test data	Cursor parameter is set to a higher value than there are items (or is a limit set), such as 500 in this case.
Test steps	1. Send **GET** `{baseUrl}/obj/deal?cursor=500` request.
Expected result	• Successful code: 200 OK. • No list is returned in the response.

Test case ID	**DA008 - Get the list of all Deals - cursor valid value**

Description	When Get the list of all Deals request has the cursor set to valid value, then results should be returned.
Precondition	Some deals exist in the system's database.
Test data	Cursor parameter is set to a valid value, such as 2 in this case.
Test steps	1. Send **GET** `{baseUrl}/obj/deal?cursor=2` request.
Expected result	• Successful code: 200 OK. • In the response, a list of items is returned starting from cursor of a valid value, such as 2 in this example.

Test case ID	**DA009 - Get the list of all Deals - cursor negative value**
Description	When Get the list of all Deals request has the cursor set to a negative value, then no results should be returned.
Precondition	Some deals exist in the system's database.
Test data	Cursor parameter is set to a negative value, such as -10 in this case.
Test steps	1. Send **GET** `{baseUrl}/obj/deal?cursor=-10` request.
Expected result	• Error code: 500 Internal Server Error. • No list is returned in the response.

Test case ID	**DA010 - Get the list of all Deals - Sort by status using a boolean**
Description	Sorting the list of all deals is possible by the status in ascending order.
Precondition	Deals exist in the system's database with one of the following statuses: • `Won` • `In progress` • `Lost`
Test data	Sort_field parameter is set to a value of status_option_status. Descending parameter is set to a value of false.
Test steps	1. Send **GET** `{baseUrl}//obj/deal?sort_field=status_option_status&descending=false` request.
Expected result	• Successful code: 200 OK. • In the response, the sorted list of items is returned in ascending order.

Test case ID	**DA011 - Find a deal with a specific title**
Description	Possible to locate and retrieve deal by title.
Precondition	Deals exist in the system's database.
Test data	Searching for a name “Pirate”, But you can choose any other to replace it.
Test steps	1. Send **GET** `{baseUrl}/obj/deal?constraints=[{ "key": "name_text", "constraint_type": "equals", "value": "Pirate" }]` request.

Expected result	• Successful code: 200 OK. • The specific deal's title is returned in the response.

Test case ID	**DA012 - Search for a deal with the non-existing title**
Description	When a non-existent deal title is used as a filter, an empty list of results is returned.
Precondition	There is no deal with the provided title.
Test data	Try using a deal title that is not in the system, such as “CRM”.
Test steps	1. Send **GET** `{baseUrl}/obj/deal?constraints=[{ "key": "name_text", "constraint_type": "equals", "value": "CRM" }]` request.
Expected result	• Successful code: 200 OK. • Empty list is returned in response.

Test case ID	**DA013 - Search for a deal with empty title**
Description	When a deal name is empty in the filter, an empty list of results is returned.
Precondition	There is a deal with empty title in the system.
Test data	-
Test steps	1. Send **GET** `{baseUrl}/obj/deal?constraints=[{"key": "name_text", "constraint_type": "is_empty"}]` request.
Expected result	• Successful code: 200 OK. • List of all deals with empty title are returned in response.

Test case ID	**DA014 - Search for a deal using a random non-existent ID**
Description	When using a non-existent deal ID as a filter, the system returns an empty list of results.
Precondition	The chosen deal ID shouldn't exist in the database.
Test data	Choose a non-existing deal ID from the system.
Test steps	1. Send **GET** `{baseUrl}/obj/deal?constraints=[{"key": "_id", "constraint_type": "equals", "value": "X0X0X0X0"}]` request
Expected result	• Error code is shown: 404 Not Found. • The user cannot retrieve any list of deals with a non-existing ID.

Test case ID	**DA015 - Search for a deal using an empty ID**
Description	When using an empty deal ID as a filter, the system returns an empty list of results.
Precondition	Some deals exist in the application.
Test data	-

Test steps	1. Send **GET** `{baseUrl}/obj/deal?constraints=[{"key": "_id", "constraint_type": "is_empty"}]` request.
Expected result	• Error code is shown: 404 Not Found. • The user cannot retrieve any deals with an empty ID.

Test case ID	**DA016 - Search for a deal using estimated value**
Description	When using an estimated value as a filter, the system returns a valid list of results.
Precondition	Some deals exist in the application.
Test data	The `"deal_value_estimation_number"` should be be greater than the minimum esitamated deal value in the system, such as 10000 in this example.
Test steps	1. Send **GET** `{baseUrl}/obj/deal?constraints=[{"key": "deal_value_estimation_number", "constraint_type":"greater than","value":"10000"}]` request.
Expected result	• Successful code: 200 OK. • In response, the list of deals consists of items with estimated values geater than the set value, such as 10000 in this example.

Test case ID	**DA017 - Search for a deal using an unique existing ID**
Description	You can find the existing deal via when searching by its unique ID.
Precondition	The chosen deal ID exists in the database.
Test data	Choose an existing deal ID from the system, such as “1625048131815x332253337528303600”.
Test steps	1. Send **GET** `{baseUrl}/obj/deal/:UniqueID` request. 2. Add an existing value, such as “1625048131815x332253337528303600”.
Expected result	• Successful code: 200 OK. • The specific deal with the entered unique ID is returned in the response.

Test case ID	**DA018 - Create a deal with min set of fields and check the created item**
Description	It is possible to create a new deal with minimum dataset, and observe if it was created successfully with given data.
Precondition	-
Test data
```
{
  "assignee__user": "1625047362532x398219546419355650",
  "company_name_text": "Team2",
  "deal_value_estimation_number": 100,
  "name_text": "Tere2",
  "order_number": 1,
  "status_option_status": "In progress",
  "visible_to_list_user": [
    "1625047362532x398219546419355650"
```
 |

	<pre>10]
	11 }</pre>

Test steps	1. Send **POST** `{baseUrl}/obj/deal` request using body from test data. 2. Copy id of the created deal from the response. 3. Send **GET** `{baseUrl}/obj/deal/:UniqueID` request with correct unique id taken from step 2.
Expected result	• Step 1: Successful code: 201 Created. Deal is created. • Step 3: Successful code: 200 OK. Correct lead is found, data corresponds to input.

Test case ID	**DA019 - Create a deal with min set of fields and no authorization**
Description	It is not possible to create a new deal without authorization.
Precondition	Authorization type is set to no authorization for this request.

Test data	<pre>1 {
	2 "assignee__user": "1625047362532x398219546419355650",
	3 "company_name_text": "Team22",
	4 "deal_value_estimation_number": 100,
	5 "name_text": "Tere22",
	6 "order_number": 1,
	7 "status_option_status": "In progress",
	8 "visible_to_list_user": [
	9 "1625047362532x398219546419355650"
	10]
	11 }</pre>

Test steps	1. Send **POST** `{baseUrl}/obj/deal` request using body from test data.
Expected result	• Error code is shown: 401 Unauthorized. • The user cannot create a new deal without authorization.

Test case ID	**DA020 - Create a deal with non-existing id**
Description	It is not possible to create a new deal with non-existing id in the id field.
Precondition	-

Test data	<pre>1 {
	2 "assignee__user": "000000000000000000000000000000000",
	3 "company_name_text": "Team22",
	4 "deal_value_estimation_number": 100,
	5 "name_text": "Tere22",
	6 "order_number": 1,
	7 "status_option_status": "In progress",
	8 "visible_to_list_user": [
	9 "1625047362532x398219546419355650"
	10]
	11 }</pre>

Test steps	1. Send **POST** `{baseUrl}/obj/deal` request using body from test data.
Expected result	• Error code is shown: 400 Bad request.

	• The user cannot create a new deal.

Test case ID	**DA021 - Create a new deal - empty body**
Description	Creating a new deal should not be possible without filling in the 'Body' field.
Precondition	-
Test data	`{` `}`
Test steps	1. Send **POST** `{baseUrl}/obj/deal` request with an empty request body as above.
Expected result	• Request should return a response with a status code of 400 bad request.

Test case ID	**DA022 - Create a new deal - empty field value for visible_to_list_user**
Description	Creating a new deal should not be possible with empty mandatory field value, such as visible_to_list_user .
Precondition	-

Test data	`{`
	`"assignee__user": "1625047362532x398219546419355650",`
	`"company_name_text": "TERE",`
	`"deal_value_estimation_number": 1,`
	`"name_text": "HO",`
	`"order_number": 1,`
	`"status_option_status": "In progress",`
	`"visible_to_list_user": []`
	`}`

Test steps	1. Send **POST** `{baseUrl}/obj/deal` using body from test data.
Expected result	• “Error code: 400 Bad Request” error message is shown. • Deal is not created.

Test case ID	**DA023 - Create a new deal - empty field value for min set of fields**
Description	Creating a new deal should not be possible with all mandatory field values being empty.
Precondition	-

Test data	`{`
	`"assignee__user": "1625047362532x398219546419355650",`
	`"company_name_text": "TERE",`
	`"deal_value_estimation_number": 1,`
	`"name_text": [],`
	`"order_number": [],`
	`"status_option_status": [],`
	`"visible_to_list_user": []`
	`}`

Test steps	1. Send **POST** `{baseUrl}/obj/deal` using body from test data.

Expected result	• “Error code: 400 Bad Request” error message is shown. • Deal is not created.

Test case ID	**DA024 - Update deal and check the data**
Description	Confirm that it is possible to update existing deal by using the 'PATCH' request.
Precondition	DA018 successfully completed.

Test data	<pre>{
	  "assignee__user": "1625051658962x581289845978797800",
	  "company_name_text": "Team2 updated",
	  "deal_value_estimation_number": 200,
	  "name_text": "Tere2 updated",
	  "order_number": 2,
	  "status_option_status": "Won",
	  "visible_to_list_user": [
	  "1657897085311x920666300518670800"
	  ]
	}</pre>

Test steps	1. Send **PATCH** `{baseUrl}/obj/deal/:UniqueID`, enter correct unique ID taken from step 2 of DA018. 2. Send **GET** `{baseUrl}/obj/deal/:UniqueID` request with correct unique id taken from previous step.
Expected result	• Step 1: Successful code: 204 No Content. Update was successful. • Step 2: Successful code: 200 OK. Correct lead is found, data corresponds to updated input.

Test case ID	**DA025 - Update deal data without authorization**
Description	Confirm that it is not possible to update existing deal data without authorization.
Precondition	DA018 successfully completed.

Test data	<pre>{
	  "company_name_text": "Team2 updated2",
	  "deal_value_estimation_number": 2000,
	  "name_text": "Tere2 updated2"
	}</pre>

Test steps	1. Send **PATCH** `{baseUrl}/obj/deal/:UniqueID`, enter correct unique ID taken from step 2 of DA018.
Expected result	• Error code: 401 Unauthorized • Error message: `"You do not have permission to modify this object"`

Test case ID	**DA026 - Update deal data with integer replaced by a string**
Description	Confirm that it is not possible to replace integer (number) by string (text) when updating the existing deal.
Precondition	DA018 successfully completed.

Test data	<pre>{
	  "deal_value_estimation_number": "letter value"
	}</pre>

Test steps	1. Send **PATCH** `{baseUrl}/obj/deal/:UniqueID`, enter correct unique ID taken from step 2 of DA018. 2. Ented the test data and replace integer with a string value for deal value estimation number.
Expected result	• Error code: 400 Bad Request. • Error message: `"Invalid data for field deal_value_estimation_number: Expected a number, but got a string (original data: \"letter value\")"`

Test case ID	**DA027 - Replace deal with empty body**
Description	Ensure that the request **does not** replace a deal when the request body is empty.
Precondition	1. The user has access to the system. 2. A deal exists in the system's database with the ID "`1625048701288x662432067036119000`"

Test data	<pre>{
	}</pre>

Test steps	1. Send **PUT** `{baseUrl}/obj/deal/:UniqueID` , Enter unique ID provided above as variable.
Expected result	• The request is not processed, and the response code '204 No Content' is displayed.

Test case ID	**DA028 - Replace deal with a new min set of data**
Description	Confirm that it is possible to replace existing minimum deal data by using the 'PUT' request.
Precondition	DA018 successfully completed.

Test data	<pre>{
	  "assignee__user": "1625051658962x581289845978797800",
	  "company_name_text": "Team2 updated2",
	  "deal_value_estimation_number": 2000,
	  "name_text": "Tere2 updated2",
	  "order_number": 22,
	  "status_option_status": "Won",
	  "visible_to_list_user": [
	  "1657897085311x920666300518670800"
	  ]
	}</pre>

Test steps	1. Send **PUT** `{baseUrl}/obj/deal/:UniqueID` , enter correct unique ID taken from DA018.

Expected result	• The request is not processed, and the response code '204 No Content' is displayed.

Test case ID	**DA029 - Delete deal without entering Unique ID**
Description	Confirm that the API appropriately manages a deletion request when the unique ID parameter is missing.
Precondition	The ID parameter is left blank.
Test data	-
Test steps	1. Send **DEL** `{baseUrl}/obj/deal/:UniqueID` , without any ID in variables.
Expected result	• A '404 Not Found' error receive as result with following message `"message": "Missing object of type deal: object with id :UniqueID does not exist"`

Notes API

Test case ID	**NA001 - Get the list of all notes**
Description	It is possible to see the full list of all existing notes
Precondition	Some notes exists in the application
Test data	-
Test steps	1. Send **GET** `{{baseURL}}/obj/note`
Expected result	• Successful code: 200 OK • List of all notes is shown in response

Test case ID	**NA002 - Get the list of all notes - limit (default to 50 required)**
Description	When Get the list of all Notes request has no limit set (it is empty), then it should show the default limit of 50 items.
Precondition	More than 50 deals exist in the system's database.
Test data	Limit parameter is chosen, but set to no value (left empty).
Test steps	1. Send **GET** `{baseUrl}/obj/note?limit=` request.
Expected result	• Successful code: 200 OK. • A default list of 50 items is returned in the response.

Test case ID	**NA003 - Get the list of all notes - cursor valid value**
Description	When Get the list of all Notes request has the cursor set to valid value, then results should be returned.
Precondition	Some notes exist in the system's database.

Test data	Cursor parameter is set to a valid value, such as 1 in this case.
Test steps	1. Send **GET** `{baseUrl}/obj/note?cursor=1` request.
Expected result	• Successful code: 200 OK. • In the response, a list of items is returned starting from cursor of a valid value, such as 1 in this example.

Test case ID	**NA004 - Get the list of all notes - sorting by content text**
Description	It is possible to get the notes by sorting content text.
Precondition	Some notes exist in the system's database.
Test data	The value of “sort_field” is marked “content_text”.
Test steps	1. Send **GET** `{{baseURL}}/obj/note?limit=50&cursor=0&sort_field=content_text`
Expected result	• Successful code: 200 OK • List of all notes is shown in alphabetical order.

Test case ID	**NA005 - Get the list of all notes - sorting by “Created By”**
Description	It is possible to get the list of notes sorted by “Created By”.
Precondition	Some notes exist in the system's database.
Test data	The value of “sort_field” is marked “Created By”.
Test steps	1. Send **GET** `{{baseURL}}/obj/note?sort_field=Created By`
Expected result	• Successful code: 200 OK • List of all notes is shown in alphabetical order.

Test case ID	**NA006 - Get the list of all notes - find a note with a specific content text**
Description	Possible to locate and retrieve note by content text.
Precondition	Notes exist in the system's database.
Test data	Searching for a note text “`blablabal`”, but you can choose any other to replace it.
Test steps	1. Send **GET** `{baseUrl}/obj/note?constraints=[{ "key": "content_text", "constraint_type": "equals", "value": "blablabal" }]` request.
Expected result	• Successful code: 200 OK. • The specific note text is returned in the response.

Test case ID	**NA007 - Get note data by unique ID**
Description	Is is possible to get the certain note data by using unique ID.
Precondition	The certain note is existing.

Test data	Unique ID of a note.
Test steps	1. Send **GET** `{{baseURL}}/obj/note/:UniqueID`
Expected result	• Successful code: 200 OK • That certain note is shown

Test case ID	**NA008 - Create a new note**
Description	It is possible to create a new note
Precondition	-
Test data	`{ "content_text": "bees" }`
Test steps	1. Send **POST** `{{baseURL}}/obj/note`
Expected result	• Successful code: 201, Success. Created one object of type note. • New note is successfully created

Test case ID	**NA009 - Create a new note - authorization**
Description	It is not possible to create a new note without authorization.
Precondition	Authorization type is set to no authorization for this request.
Test data	`{ "content_text": "bees" }`
Test steps	1. Send **POST** `{{baseURL}}/obj/note`
Expected result	• Error code is shown: 401 Unauthorized. • The user cannot create a new deal without authorization.

Test case ID	**NA010 - Create a new note with empty body**
Description	It is not possible to create a new note with no text.
Precondition	-
Test data	`{ "content_text": [] }`
Test steps	1. Send **POST** `{{baseURL}}/obj/note`
Expected result	• Request should return a response with a status code of 400 bad request. • New note is not created.

Test case ID	**NA011 - Modify note and check the data**

Description	It is possible to modify note data by using unique ID, and check the new data.
Precondition	Note is existing.

Test data	<pre>{
	  "content_text": "beesbees 111"
	}</pre>

Test steps	1. Send **PATCH** `{{baseURL}}/obj/note/:UniqueID`, write unique ID taken from NA008 result. 2. Seng **GET** `{{baseURL}}/obj/note/:UniqueID` ,write unique ID taken from TCN008 result.
Expected result	Step 1: Successful code: 204 No Content, note updated. Step 2: Successful code: 200 OK, correct note is found.

Test case ID	**NA012 - Modify note with no authorization**
Description	It is not possible to modify a note without authorization.
Precondition	Authorization type is set to no authorization for this request.

Test data	<pre>{
	  "content_text": "beesbeesbees"
	}</pre>

Test steps	1. Send **PATCH** `{{baseURL}}/obj/note/:UniqueID`, write unique ID taken from TCN008 result.
Expected result	• Error code is shown: 401 Unauthorized. • The user cannot create a new deal without authorization.

Test case ID	**NA013 - Modify note of non-existent ID**
Description	It is not possible to modify note data of non-existent ID.
Precondition	Note is not existing, e.g. use ID “11111111111111111111111111111111” in our example.

Test data	<pre>{
	  "content_text": "beesbeesbees"
	}</pre>

Test steps	1. Send **PATCH** `{{baseURL}}/obj/note/:UniqueID` with unique ID that does not exist, e.g. taken from precondition above in this example.
Expected result	• A '404 Not Found' error received as result with following message: `"Missing object of type note: object with id 1111111111111111111111111111 does not exist"`

Test case ID	**NA014 - Replace note and check the data**
Description	It is possible to replace and check the note’s data

Precondition	Note is existing

Test data	<pre>{
	  "content_text": "BEES"
	}</pre>

Test steps	1. Send **PUT** `{{baseURL}}/obj/note/:UniqueID`, write unique ID taken from NA011 result. 2. Send **GET** `{{baseURL}}/obj/note/:UniqueID`, write unique ID taken from NA011 result.
Expected result	Step 1: Successful code: 204 No Content, note replaced. Step 2: Successful code: 200 OK, correct note is found.

Test case ID	**NA015 - Delete note and check the data**
Description	It is possible to delete and check if this note is gone.
Precondition	Note is existing.
Test data	-
Test steps	1. Send **DELETE** `{{baseURL}}/obj/note/:UniqueID`, write unique ID taken from TCN011 result. 2. Send **GET** `{{baseURL}}/obj/note/:UniqueID`, write unique ID taken from TCN011 result.
Expected result	Step 1: Successful code: 204 No Content, delete was successful. Step 2: Error code: 404 Not Found, `"status": "MISSING_DATA", "message": "Missing object of type lead: object with id {{UniqueId}} does not exist"`