Team 5 2. Test cases

Deals API

GET test cases (Kristin)

Test case ID	**DA001 - GET the list of all deals**
Description	Make sure it is possible to retrieve the full list of all existing deals
Precondition	Some deals exists in the application
Test data	-
Test steps	1. Send **GET** `{baseUrl}/obj/deal`
Expected result	• Successful code: 200 OK is shown • List of all deals is returned in response

Test case ID	**DA002 - GET the list of all deals - Get the list of all deals without authorization**
Description	Verify that when trying to retrieve the full list of all existing deals as an unauthorised user, an error message is sent and access to the list of deals is denied
Precondition	Some deals exists in the application
Test data	-
Test steps	1. Send **GET** `{baseUrl}/obj/deal` 2. Don't include the mandatory authorization code
Expected result	• Error code: 401 Permission denied should be shown • Unauthorised user should not have access to the deals list

Test case ID	**DA003 - GET the list of all deals - Sorting deals list with valid field**
Description	Verify that it is possible to sort the list by applying a field `assignee__user`
Precondition	Some deals exists in the application
Test data	-
Test steps	1. Send **GET** `{baseUrl}/obj/deal?sort_field=assignee__user`
Expected result	• Successful code: 200 OK is shown • List of all deals sorted by assignee user is returned in response

Test case ID	**DA004 - GET the list of all deals - Sorting deals list with invalid field**

Description	Verify that when trying to sort the list with an invalid field, an error message is shown in response
Precondition	Some deals exists in the application
Test data	-
Test steps	1. Send **GET** `{baseUrl}/obj/deal?sort_field=namename`
Expected result	• Error code: 404 Not Found • Error message is shown in response: `{"statusCode":404,"body": {"status":"NOT_FOUND","message":"Field not found namename for type deal"}}`

Test case ID	**DA005 - GET the list of all deals - Get deals list with default limit (50)**
Description	Verify that when applying the limit value as 50, it returns a list with the expected number of deals
Precondition	At least 50+ deals exists in the application
Test data	-
Test steps	1. Send **GET** `{baseUrl}/obj/deal?limit=50`
Expected result	• Successful code: 200 OK is shown • The response list contains 50 deals

Test case ID	**DA006 - GET the list of all deals - Get deals list with negative limit**
Description	Verify that when applying the limit value as -1, an error message is shown in response
Precondition	Some deals exists in the application
Test data	-
Test steps	1. Send **GET** `{baseUrl}/obj/deal?limit=-1`
Expected result	• Error code: 400 Database retrieval failure should be shown • Limit value can't be a negative number

Test case ID	**DA007 - GET the list of all deals - Get deals list with UniqueID**
Description	Verify that when searching a deal with its UniqueID, it is returned with the right response
Precondition	Some deals exists in the application and a deal with an UniqueID exists that is searched
Test data	-
Test steps	1. Send **GET** `{baseUrl}/obj/deal/:UniqueID`
Expected result	• Successful code: 200 OK is shown • The right deal with the UniqueID that was applied is shown in response

Test case ID	DA008 - GET the list of all deals - Sorting deals list by deal status
Description	Verify that it is possible to sort deals by their status
Precondition	Three types of deals exist in the application: • Won • Lost • In progress
Test data	-
Test steps	1. Send **GET** `{baseUrl}/obj/deal?constraints=[{"key": "status_option_status", "constraint_type": "equals", "value": "Won" }]`
Expected result	• Successful code: 200 OK is shown • Sorted deals list with the right status are returned in response

Test case ID	DA009 - GET the list of all deals - Sorting deals list by non-existing status
Description	Verify that it is possible to sort deals by their status
Precondition	Three types of deals exist in the application: • Won • Lost • In progress
Test data	-
Test steps	1. Send **GET** `{baseUrl}/obj/deal?constraints=[{"key": "status_option_status", "constraint_type": "equals", "value": "In sorting" }]`
Expected result	• Error code: 400 Bad Request is shown • Error message is shown in response: `"Invalid data for endpoint deal, key status_option_status: could not parse this as a Status"`

POST + GET (Kristin)

Test case ID	DA010 - POST + GET - Create a new deal and get the deal using it's UniqueID
Description	Verify that it is possible to create a new deal and observe if it is was created successfully with given data
Precondition	-
Test data
```
{
  "assignee__user": "1625047362532x398219546419355650",
  "company_name_text": "Kris Company",
  "deal_value_estimation_number": 3500,
  "name_text": "Team5",
  "order_number": 1,
  "status_option_status": "In progress",
```
 |

	<pre>"visible_to_list_user": ["1625047362532x398219546419355650"] }</pre>
Test steps	1. Send **POST** `{baseUrl}/obj/deal` request using body from test data 2. Copy the UniqueID of the created deal 3. Send **GET** `{baseUrl}/obj/deal/:UniqueID` using the correct UniqueID copied from step 2
Expected result	• Step 1: Successful code: 201 Created is shown in response and the deal is created • Step 3: Successful code: 200 OK, correct deal is found

DELETE + GET (Kristin)

Test case ID	**DA011 - DELETE + GET - Delete deal and check data**
Description	Verify that it is possible to delete existing deal
Precondition	Deal created <pre>{ "assignee__user": "1625047362532x398219546419355650", "company_name_text": "Kris Company for deletion", "deal_value_estimation_number": 3500, "name_text": "Team5", "order_number": 1, "status_option_status": "In progress", "visible_to_list_user": ["1625047362532x398219546419355650"] }</pre>
Test data	-
Test steps	1. Send **DELETE** `{baseUrl}/obj/deal/:UniqueID` using the correct UniqueID from deal shown in preconditions 2. Send **GET** `{baseUrl}/obj/deal/:UniqueID` using the same UniqueID (as from previous step)
Expected result	• Step 1: Successful code: 204 No Content, showing that delete was successful • Step 2: Error message is shown in response: `"Missing object of type deal: object with id 1695911376675x958509358206083700 does not exist"`

POST test cases (Vitali)

Test case ID	**POST001 - Create a new deal**
Description	It is possible to create a new deal with all required fields filled in

Precondition	You have authorization token and user id
Test data	<pre>{ "assignee__user": "1625051658962x581289845978797800", "company_name_text": "Team5_Vitali001", "deal_value_estimation_number": 12500, "name_text": "Team5_Vitali_nametext", "order_number": 999, "status_option_status": "In progress", "visible_to_list_user": ["1625051658962x581289845978797800", "1625047373081x778685405315052900"] }</pre>
Test steps	1. Send POST `{baseURL}/obj/deal` using code from test data and your chosen user id 2. Copy the new deal ID when it's created 3. Send GET `{baseURL}/obj/deal:UniqueID` using the ID copied in previous step to confirm the information is saved correctly.
Expected result	• the POST request went through successfully with "201 Success" code and the output provided the ID of a new deal • the GET request with the ID of a newly created deal outputs the information of this specific deal confirming the info has been saved

Test case ID	**POST002 - Create a new deal - no auth**
Description	It is not possible to create a new deal with no authorization token
Precondition	set authorization to "no auth"
Test data	<pre>{ "assignee__user": "1625051658962x581289845978797800", "company_name_text": "Team5_Vitali001", "deal_value_estimation_number": 12500, "name_text": "Team5_Vitali_nametext", "order_number": 999, "status_option_status": "In progress", "visible_to_list_user": ["1625051658962x581289845978797800", "1625047373081x778685405315052900"] }</pre>
Test steps	1. Send POST `{baseURL}/obj/deal` using code from test data and your chosen user id
Expected result	• You get the "401 Unauthorized" code and an output error messae states: `"error_class": "Unauthorized" and "translation": "Permission denied: cannot create this object"`

Test case ID	**POST003 - Create a new deal with no info - request body is empty**

Description	It is not possible to create a new deal with no information filled in the body of the request
Precondition	You have authorization token
Test data	```1 ```
Test steps	1. Send POST `{baseURL}/obj/deal` with the body of the request empty
Expected result	• You are getting the "400 Bad request" status code. The output message should be stating that some info is missing.

Test case ID	**POST004 - Create a new deal with non-existent user**
Description	It is not possible to create a new deal with a random user UniqueID (non-existent user)
Precondition	You have authorization token
Test data	```1 { 2 "assignee__user": "0001112223334x445556667778889990", 3 "company_name_text": "Team5_Vitali001", 4 "deal_value_estimation_number": 12500, 5 "description_text": "Description text 001", 6 "funnel_custom_funnel1": "1678718042244x598870217126314000", 7 "lead_custom_lead": "1694453584971x281671486172663000", 8 "name_text": "Team5_Vitali_nametext", 9 "order_number": 999, 10 "stage_custom_stage": "1625047818390x664413796313397600", 11 "status_option_status": "In progress", 12 "visible_to_list_user": [13 "1625051658962x581289845978797800", 14 "1625047373081x778685405315052900" 15] 16 }```
Test steps	1. Send POST `{baseURL}/obj/deal` using code from test data and a random user UniqueID
Expected result	• you get the "400 Bad Request" status code and an error message: ```1 { 2 "statusCode": 400, 3 "body": { 4 "status": "MISSING_DATA", 5 "message": "Invalid data for field assignee__user: objec 6 } 7 }```

PUT test cases (Vitali)

Test case ID	**PUT001 - Replace deal data**
Description	It is possible to replace the deal data
Precondition	Some deals must exist; you have authorization token

Test data	<pre>{ "assignee__user": "1625051658962x581289845978797800", "company_name_text": "Team5_Vitali001", "deal_value_estimation_number": 256256, "description_text": "Description text 001", "funnel_custom_funnel1": "1678718042244x598870217126314000", "lead_custom_lead": "1694453584971x281671486172663000", "name_text": "Team5_Vitali_nametext", "order_number": 998, "stage_custom_stage": "1625047818390x664413796313397600", "status_option_status": "In progress", "visible_to_list_user": ["1625051658962x581289845978797800"] }</pre>
Test steps	1. Send **PUT** `{baseURL}/obj/deal/:UniqueID` using the UniqueID of a deal created in test case POST001. Use code from test data in the body of the request as an example and change some info. 2. Send **GET** `{baseURL}/obj/deal:UniqueID` using the same UniqueID to confirm the information is replaced correctly.
Expected result	• You get the “204 Success” status code. • The **GET** request with the same UniqueID returns the deal with information replaced.

Test case ID	**PUT002 - Replace deal data with no authorization**
Description	It is not possible to replace the deal data when not authorized
Precondition	Some deals must exist; the autorization must be set to “no auth”
Test data	<pre>{ "assignee__user": "1625051658962x581289845978797800", "company_name_text": "Team5_Vitali001", "deal_value_estimation_number": 256256, "description_text": "Description text 001", "funnel_custom_funnel1": "1678718042244x598870217126314000", "lead_custom_lead": "1694453584971x281671486172663000", "name_text": "Team5_Vitali_nametext", "order_number": 998, "stage_custom_stage": "1625047818390x664413796313397600", "status_option_status": "In progress", "visible_to_list_user": ["1625051658962x581289845978797800"] }</pre>
Test steps	1. Send **PUT** `{baseURL}/obj/deal/:UniqueID` using the UniqueID of a deal created in test case POST001. Use code from test data in the body of the request as an example and change some info.
Expected result	• You get the “401 Unauthorized” code and an output error messae states: `"error_class": "Unauthorized" and "translation": "You do not have permission to modify this object"`

Test case ID	**PUT003 - Replace deal data with no info - empty request body**
Description	It is not possible to replace the deal information with nothing, i.e. leaving the request body empty
Precondition	Some deals must exist; you have authorization token
Test data	`1`
Test steps	1. Send **PUT** `{baseURL}/obj/deal/:UniqueID` using the UniqueID of a deal created in test case POST001 and leaving the request body empty.
Expected result	• You are getting the "400 Bad request" status code. The output message should be stating that some info is missing.

PATCH test cases (Vitali)

Test case ID	**PATCH001 - Modify deal data**
Description	It is possible to modify the deal info
Precondition	You have authorization token
Test data	`{ "assignee__user": "1625051658962x581289845978797800", "company_name_text": "Team5_Vitali001", "deal_value_estimation_number": 12500, "description_text": "Description text 001", "funnel_custom_funnel1": "1678718042244x598870217126314000", "lead_custom_lead": "1694453584971x281671486172663000", "name_text": "Team5_Vitali_nametext", "order_number": 999, "stage_custom_stage": "1625047818390x664413796313397600", "status_option_status": "In progress", "visible_to_list_user": ["1625051658962x581289845978797800", "1625047373081x778685405315052900"] }`
Test steps	1. Send **PATCH** `{baseURL}/obj/deal/:UniqueID` using UniqueID of a lead created in test case POST001. 2. Use code form test data as a basis. Make changes to some values and remove all others that are not intended to be modified. 3. Send **GET** `{baseURL}/obj/deal:UniqueID` using the same UniqueID to confirm the information is modified correctly.
Expected result	• After sending the **PATCH** request you get the status code “204 Success” • The **GET** request with the same UniqueID returns the deal with correctly modified information.

Test case ID	**PATCH002 - Modify deal data - no auth**
Description	It is not possible to modify the deal data with no authorization

Precondition	The authorization is set to “no auth”
Test data	<pre>{ "deal_value_estimation_number": 9999, "description_text": "Modified text with PATCH", "order_number": 256, "status_option_status": "Won" }</pre>
Test steps	1. Send **PATCH** `{baseURL}/obj/deal/:UniqueID` using UniqueID of a lead created in test case POST001.
Expected result	• You get the “401 Unauthorized” code and an output error messae states: `"error_class": "Unauthorized"` and `"translation": "You do not have permission to modify this object"`

Test case ID	**PATCH003 - Modify deal data - no values in fields**
Description	It is not possible to modify deal data into empty values
Precondition	You have authorization token
Test data	<pre>{ "deal_value_estimation_number": , "description_text": "", "order_number": , "status_option_status": "" }</pre>
Test steps	1. Send **PATCH** `{baseURL}/obj/deal/:UniqueID` using UniqueID of a lead created in test case POST001. 2. Use the code from test data. Note that there is no values entered.
Expected result	• Attempting to send the **PATCH** request outputs the status code “400 Bad Request” with the message `Error: Error parsing request body: Unexpected token , in JSON at position xx`

Test case ID	**PATCH004 - Modify deal data - integers instead of strings**
Description	It is not possible to use integers instead of strings
Precondition	You have authorization token
Test data	<pre>{ "company_name_text": "Team5_Vitali001", "description_text": "Description text 001", "name_text": "Team5_Vitali_nametext", }</pre>
Test steps	1. Send **PATCH** `{baseURL}/obj/deal/:UniqueID` using UniqueID of a lead created in test case POST001. 2. Use the code from test data. Enter integers (numbers only, no quotes) instead of string values.

Expected result	• Attempting to send the **PATCH** request outputs the status code “400 Bad Request” with the message: `{ "statusCode": 400, "body": { "status": "INVALID_DATA", "message": "Invalid data for field company_name_text: Ex } }`

Notes API

GET test cases (Ivan)

Test case ID	**NA001 - GET the list of all notes**
Description	Varify that API returns 200 OK status code and provides a list containing all the notes.
Precondition	The user is authenticated, and multiple notes exist.
Test data	-
Test steps	1. Send **GET** `{baseUrl}/obj/note`
Expected result	The API should return a 200 OK status code and provide a list containing all the notes.

Test case ID	**NA002 GET note Data by Unique ID**
Description	Ensure that the API allows retrieving the data of a specific note by its unique ID.
Precondition	The user is authenticated, and a note exists.
Test data	
Test steps	1. Send **GET** `{baseUrl}/obj/note/:UniqueID` with UniqueID obtained in test case **NA007** and use code from test data in the body of the request.
Expected result	1. The API must return the data of a note by its unique Id.

Test case ID	**NA003 Read a List of All Notes and Sort by Creation Date**
Description	Ensure that the API allows retrieving a list of notes sorted by their creation dates.
Precondition	The user is authenticated
Test data	-
Test steps	1. Send **GET** `{baseUrl}/obj/note/?sort_field=Created Date`
Expected result	The API should return a list of all notes sorted by the creation date.

Test case ID	**NA005 Unauthorized Access**
Description	Ensure that the API restricts unauthorized access to note data retrieval.
Precondition	The user is not authenticated.
Test data	
Test steps	1. Send **GET** `{baseUrl}/obj/note`
Expected result	The API should return a 401 Permission denied.

POST + GET (Ivan)

Test case ID	**NA007 Create a New Note and check the created object.**
Description	Verify that a new note can be successfully created through the API.
Precondition	The user is authenticated.
Test data	<pre>{ "content_text": "This is test for Dobby" }</pre>
Test steps	1. Send a **POST** `{baseUrl}/obj/note` request using the code from test data in the body section. 2. Verify that the API returns a 201 Created status code. 3. Copy id of the created lead from the response 4. Send **GET** `{baseUrl}/obj/note/:UniqueID`, set correct UniqueID (taken from step 2)
Expected result	Step 1: **POST** request is sent. Step 2: Successful code: 201 Success. Created one object of type note. is displayed. Step 3: ID can be copied. Step 4. **GET** request returns previeously created object with valid data.

Test case ID	**NA008 Attempt to Create a Note with Missing `content_text`.**
Description	Verify that the API handles the scenario when a note is created with a missing `content_text` and if the POST request is successfull check if the test data of the created object is correct.
Precondition	The user is authenticated.
Test data	<pre>{ "content_text": "" }</pre>
Test steps	1. Send a **POST** `{baseUrl}/obj/note` request to create a new note with missing content_text .

Expected result	1. The API should return a 400 Bad Request status code.

Test case ID	**NA009 - Create a new note without authorization.**
Description	Verify that a new note cannot be created by an unauthorized user.
Precondition	User is not authenticated
Test data	`{ "content_text": "There is no Dumledore" }`
Test steps	1. Send **POST** `{baseUrl}/obj/note` request to the API endpoint without authentication.
Expected result	1. The API should return an error indicating unauthorized access with code 401 Permission denied.

PATCH + GET (Ivan)

Test case ID	**NA010 - Modify existing note**
Description	Ensure that the API allows updating of note data.
Precondition	User is authorized and the note exists.
Test data	`{ "content_text": "Tom Riddle is back" }`
Test steps	1. Send **PATCH** `{baseUrl}/obj/note/:UniqueID`, enter UniqueID obtained in test case **NA007** and use code from test data in the body of the request. 2. Send **GET** `{baseUrl}/obj/note/:UniqueID` using the same UniqueID to check the result
Expected result	1. The API should return a 204 code. 2. The **GET** request outputs the note with the successfully modified data

PATCH (Ivan)

Test case ID	**NA011 - Modify a note that doesn't exist.**
Description	Verify that the API handles the scenario when a user is trying to modify a note that doesn't exist.
Precondition	User is authorized and the note does not exist.
Test data	`{ "content_text": "Peter Pettigrew is no real"`

	`3` `}`
Test steps	1. Send **PATCH** `{baseUrl}/obj/note/:UniqueID`, enter made up UniqueID.
Expected result	1. **PATCH** request is denied and 404 error code is displayed.

Test case ID	**NA012 - Modify a note without body content.**
Description	Verify that the API handles the scenario when a note is modified without body section filled.
Precondition	User is authorized and the note exists.
Test data	`1`
Test steps	1. Send **PATCH** `{baseUrl}/obj/note/:UniqueID`, enter UniqueID provided in the test data enter UniqueID obtained in test case **NA007** and use code from test data in the body of the request.
Expected result	1. **PATCH** request is denied and 400 error code is displayed. Database retrieval failure.

PUT + GET (Andre)

Test case ID	**NPUT001 Replace already existing note**
Description	Verify if you can replace an already existing note
Precondition	Authorization is successful and a note exists
Test data	`1` `{` `2` `"content_text": "TEAM5"` `3` `}`
Test steps	1. Send **PUT** `{baseUrl}/obj/note/:UniqueID`, enter UniqueID obtained in test case **NA007** and use code from test data in the body of the request. 2. Sent **GET** `{baseUrl}/obj/note/:UniqueID`, use the same UniqueID as in step 1.
Expected result	1. Successful code: 204 No Content. Meaning the reaplacement of the note data has been successful. 2. The **GET** request outputs the note by the same UniqueID showing the successfully replaced data.

Test case ID	**NPUT002 Replace already existing note without authorization**
Description	User can't replace note with no authorization
Precondition	No authorization is used
Test data	`1` `{` `2` `"content_text": "TEAM5"` `3` `}`

Test steps	1. Send **PUT** `{baseUrl}/obj/note/:UniqueID`, enter UniqueID obtained in test case **NA007** and use code from test data in the body of the request.
Expected result	1. You get status code “401 Unauthorized”, `"You do not have permission to modify this object"`

DELETE + GET (Andre)

Test case ID	**NDELETE001 Delete note**
Description	It is possible to delete
Precondition	The user is authorized and has valid authentication credentials
Test data	-
Test steps	1. Send **DELETE** `{{baseURL}}/obj/note/:UniqueID`, enter UniqueID obtained in test case **NA007** and use code from test data in the body of the request. 2. Send **GET** `{{baseURL}}/obj/note/:UniqueID`, us the same UniqueID
Expected result	1. Successful code: 204 No Content. DELETED 2. Successful code: Error code: 404 Not Found

Test case ID	**NDELETE002 Can’t delete note without authorization**
Description	It is not possible to delete if user is not authorized
Precondition	The user is not authorized and can’t delete.
Test data	-
Test steps	1. Send **DELETE** `{{baseURL}}/obj/note/:UniqueID`, enter UniqueID obtained in test case **NA007** and use code from test data in the body of the request.
Expected result	1. Successful code: 401 Unauthorized, `"You do not have permission to delete this object"`