NextCloud Functionl Tests

Release 1.0

Skander Lahbaiel

Apr 17, 2024

CONTENTS:

CHAPTER

ONE

FUNCTIONAL_TESTING

1.1 cleanup package

1.1.1 Submodules

1.1.2 cleanup.delete_test_files module

`cleanup.delete_test_files.`**`delete_files_starting_with`**(*prefix*, *directory*)

This module provides utility functions to delete test-related files from specified directories. It primarily supports cleanup activities post-testing by removing files that match a given prefix. This is particularly useful in scenarios where test execution generates temporary files that need to be cleaned up afterwards.

The module relies on environment variables to dynamically set the paths of directories involved in the testing processes, such as where downloaded files or created files during tests are stored. Users must configure these environment variables appropriately to ensure that the cleanup process targets the correct directories.

Environment Variables:

The module uses the following environment variables to determine the directories for file cleanup operations:

- `DOWNLOAD_DIRECTORY`: Specifies the directory path where files downloaded during tests are stored. This path is used by default if no other directory is specified in the cleanup function calls.
- `CREATED_FILES_PATH`: Specifies the directory path where files created during tests are stored. This allows for separation of download and upload test artifacts.

Setting Environment Variables:

To set these environment variables in your environment, follow the instructions based on your operating system:

For Unix/Linux/macOS:

Open your terminal and use the `export` command to set each environment variable. For example:

```
export DOWNLOAD_DIRECTORY=/path/to/download/directory
export CREATED_FILES_PATH=/path/to/created/files/directory
```

These commands can be added to your shell profile file (e.g., `.bashrc`, `.zshrc`) to persist the variables across sessions.

For Windows:

Use the System Properties dialog to set environment variables:

1. Open System Properties (you can search for it in the Start menu).
2. Go to the Advanced tab and click on Environment Variables.
3. Add new system variables for `DOWNLOAD_DIRECTORY` and `CREATED_FILES_PATH` with their respective paths.

Alternatively, set them temporarily in a Command Prompt window:

```
set DOWNLOAD_DIRECTORY=C:\path\to\download\directory
set CREATED_FILES_PATH=C:\path\to\created\files\directory
```

Functions:

The module contains the following function(s):

- `delete_files_starting_with(prefix, directory)`: Deletes files in the specified `directory` that start with the given `prefix`. By default, it targets the `DOWNLOAD_DIRECTORY`. The function prints the names of deleted files or indicates if no files were deleted.

Usage:

To use this module, import it into your test scripts and invoke `delete_files_starting_with` with the appropriate arguments. Here's an example that cleans up test-related files from both download and upload directories:

```
from functional_testing.cleanup import delete_files_starting_with, DOWNLOAD_
↪DIRECTORY, CREATED_FILES_PATH

# Delete downloaded test files
delete_files_starting_with('test_', DOWNLOAD_DIRECTORY)

# Delete uploaded/created test files
delete_files_starting_with('test_', CREATED_FILES_PATH)
```

Ensure you have set the necessary environment variables as described to correctly target the cleanup directories.

Note: This module assumes that the specified directories exist and that the executing user has the necessary permissions to delete files from them.

`cleanup.delete_test_files.`**`main`**`()`

The main function that executes the cleanup process.

This function calls *delete_files_starting_with* for both the download directory and the directory where created files are stored, using the prefix `'test_'` to identify files targeted for deletion.

The main function is necessary because this script is intended to be executed directly, specifically from a setup.sh file which automates the testing process. By defining a main function, we ensure that the cleanup process is executed when the script is run as a standalone program. This is controlled by the *if __name__ == "__main__":* condition at the end of the script.

The main function is a common pattern in Python programming for providing a script entry point. It allows the script to act as either reusable modules or as standalone programs.

1.1.3 Module contents

1.2 config package

1.2.1 Submodules

1.2.2 config.configuration module

This module defines configurations and locators for Selenium-based functional testing of the NextCloud application.

The configurations include paths, URLs, authentication details, file paths and sizes, base names and extensions for test files, and XPath locators for various UI elements necessary for user management, file upload/download, and user session management.

Environment Variables:

The module leverages environment variables to allow flexible configuration across different deployment environments. Users should set the following environment variables in their testing environment before executing tests:

- *BASE_DIR*: The base directory path where created files and downloads during tests will be stored.
- *CHROME_DRIVER_PATH*: The file system path to the ChromeDriver executable. Defaults to */usr/bin/chromedriver* if not set.
- *TARGET_URL*: The URL of the NextCloud instance to be tested. Example: 'https://nextcloud.example.com'
- *USERNAME*: The username for authenticating with the NextCloud instance. Defaults to 'admin'.
- *PASSWORD*: The password for the user. Example: 'password123'.
- *FILE_SIZE_MEDIUM*: Specifies the size in mega Bytes of medium files in tests. Defaults to '1'.
- *FILE_SIZE_LARGE*: Specifies the size in mega Bytes of large files in tests. Defaults to '1'.
- *TEST_FILE_BASE_LARGE*: The base name for large test files. Defaults to 'test_large_file'.
- *TEST_FILE_BASE_MEDIUM*: The base name for medium test files. Defaults to 'test_medium_file'.
- *EXTENSION*: The file extension for test files. Defaults to '.txt'.

Setting Environment Variables

Environment variables can be set in various ways depending on the operating system being used. This project contains a script that sets the environment variables for you. But if you want to set them manually, follow the instructions below:

Unix-like systems (Linux/macOS)

You can export the variables in your shell before running the tests:

```
export BASE_DIR=/path/to/base/dir
export CHROME_DRIVER_PATH=/path/to/chromedriver
export TARGET_URL=https://nextcloud.example.com
export USERNAME=admin
export PASSWORD=secret
```

Windows systems

You can set environment variables using the 'set' command in Command Prompt:

```
set BASE_DIR=C:\path\to\base\dir
set CHROME_DRIVER_PATH=C:\path\to\chromedriver.exe
set TARGET_URL=https://nextcloud.example.com
set USERNAME=admin
set PASSWORD=secret
```

Usage

This module is intended to be imported and used in functional test scripts for NextCloud. The configurations and locators defined herein facilitate interaction with the NextCloud UI through Selenium WebDriver for various testing scenarios, such as user management, file operations, and session management.

Example usage in a test script:

```
from selenium import webdriver
import functional_test_config as config

driver = webdriver.Chrome(config.CHROME_DRIVER_PATH)
driver.get(config.TARGET_URL)
```

Note: Ensure that the ChromeDriver version matches the version of Chrome installed on the testing system.

XPath Locators

This module defines several XPath locators for interacting with the NextCloud UI. These locators are used to find and interact with various buttons and other UI elements. These XPaths should work fine, until major updates in the NextCloud UI occur.

- *LOGIN_BUTTON_XPATH*: The XPath locator for the login button. Example: *'//input[@type="submit"]'*.
- *LOGOUT_BUTTON_XPATH*: The XPath locator for the logout button. Example: *'//a[@id="logout"]'*.
- *UPLOAD_BUTTON_XPATH*: The XPath locator for the file upload button. Example: *'//input[@type="file"]'*.
- *DELETE_BUTTON_XPATH*: The XPath locator for the delete file button. Example: *'//a[@class="action delete"]'*.

These XPath locators are used in conjunction with Selenium's *find_element_by_xpath* method to find and interact with the corresponding UI elements. For example:

```
login_button = driver.find_element_by_xpath(config.LOGIN_BUTTON_XPATH)
login_button.click()
```

Note: The exact XPath locators will depend on the specific structure and styling of your NextCloud instance's UI. If the UI changes, these locators may need to be updated.

1.2.3 Module contents

1.3 file_management_cycle module

1.3.1 Automated Test Sequence for File and User Management Functionalities

This automated test script is designed to validate the comprehensive functionalities related to file and user management within a web application environment. It executes a series of operations to test the system's ability to handle user and file lifecycle processes, such as adding and removing users, uploading, downloading, sharing files, and verifying file integrity, within a single, continuous execution flow.

Note: This script is intended to be executed indirectly via the `setup.sh` shell script after all necessary environment variables have been configured. It is an example of how the test modules can be utilized in a real-world scenario, demonstrating the critical nature of the tested functionalities.

The test script follows a predetermined sequence of steps *explained below* to simulate realistic application usage scenarios. The steps are encapsulated in a function.

Execution Context

- **Not for Direct Execution**: This script is executed as part of the `setup.sh` script, which prepares the environment and triggers this test sequence.
- **Environment Configuration**: Necessary environment variables and application settings must be configured prior to execution.

1.3.2 Example Usage

This script is not intended for direct call. It is automatically executed, after setting up all environment variables through `setup.sh`:

```
./setup.sh `'username'` `'password'` # This script indirectly executes the Python test␣
↪script.
```

Note:

- The test utilizes pytest fixtures for setting up the driver and test files.
- Detailed error messages are generated for failures, facilitating quick identification and remediation of issues encountered during testing.

The stringent test sequence and its execution context highlight the script's role in ensuring the application's operational integrity and reliability.

`file_management_cycle.`**`driver`**`()`

Pytest fixture that sets up and tears down a webdriver session.

This fixture uses the *setup_driver* function to initialize a webdriver session. The driver object is then yielded for use in other test functions.

After all tests have finished using the driver, the *driver.quit* method is called to close the browser and end the webdriver session.

The fixture has a session scope, which means it's invoked once per test session. The same driver object can be used across multiple test functions in the test session.

Returns
A webdriver object for interacting with the browser.

Return type
webdriver object

`file_management_cycle.`**`test_file_management_cycle`**`(`*driver*, *test_files*`)`

Test the file management cycle: upload, download, check integrity, share, delete.

This function tests the entire file management cycle in a sequential manner. It includes the following steps:

1.3.3 Workflow

1. **Login**
2. **Add a User**
3. **Upload Medium and Large Files**
4. **Download Files**
5. **Verify File Integrity**
6. **Share File**
7. **Delete Files**
8. **Delete User**
9. **Logout**

Warning: If any step in this sequence fails, the entire test is considered failed and an assertion error is raised with a relevant error message. This strict pass/fail criteria ensure the application's critical functionalities are robustly validated. The interdependency of test steps highlights the integrated nature of the application's functionalities.

param driver
A webdriver object for interacting with the browser.

type driver
webdriver object

param test_files
A tuple containing the paths and names of the medium and large size test files.

type test_files
tuple

`file_management_cycle.`**`test_files`**`()`

Pytest fixture that creates and returns paths and names of medium and large size test files.

This fixture uses the *create_testing_files* function to create medium and large size test files. The paths and names of these files are then returned for use in other test functions.

The fixture has a session scope, which means it's invoked once per test session. The created files can be used across multiple test functions in the test session.

Returns
A tuple containing the paths and names of the medium and large size test files. The tuple is structured as follows: (medium_size_file_path, medium_size_file_name, large_size_file_path, large_size_file_name)

Return type
tuple

1.4 tests package

1.4.1 Submodules

1.4.2 tests.end_to_end_encryption module

Testing End-to-End Encryption (E2EE) in Nextcloud Client

This documentation provides a comprehensive guide to manually test the End-to-End Encryption (E2EE) functionality in the Nextcloud client. E2EE is designed to protect user data from access by the server or any other unauthorized entities. Due to the complexity of interacting with the GUI in a Linux environment, this test is conducted manually.

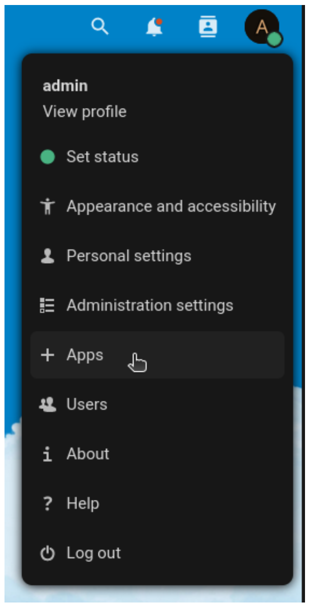
Prerequisites

Before proceeding with the testing, ensure the following prerequisites are met:

- A functioning Nextcloud server deployment.
- Access to a Linux environment with GUI support for installing and operating the Nextcloud client.
- Administrative access to the Kubernetes cluster where Nextcloud is deployed, to verify encrypted files directly on the server.

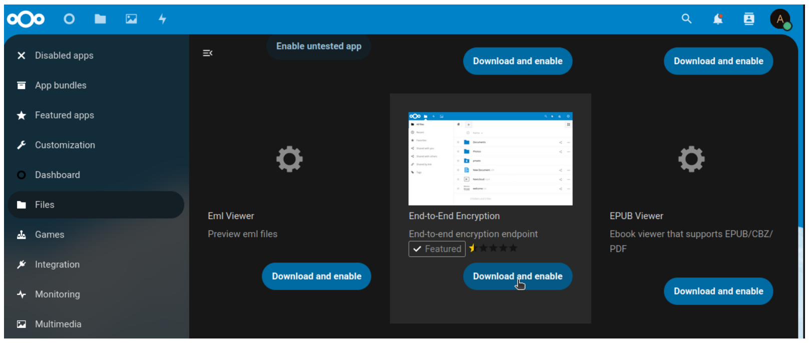
Steps

1. **Install and Enable the End-to-End Encryption App**: Before installing the Nextcloud client, ensure the End-to-End Encryption (E2EE) app is installed and enabled on your Nextcloud server. This app is crucial for securing your files during storage and transfer.
    - **Navigate to Apps**: Log into your Nextcloud server web interface as an administrator.



    - **Access the Apps Management Page**: Click on your user profile icon at the top right corner of the interface and select 'Apps' from the dropdown menu.

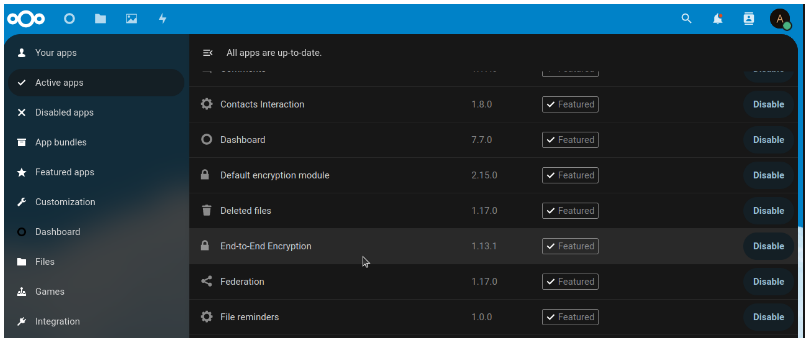
- **Install the End-to-End Encryption App**: In the Apps management page, go to the 'Files' category. Find the 'End-to-End Encryption' app in the list, and click 'Download and enable'.



- **Enable the App**: After the download completes, the app will automatically be enabled. Verify this by checking that it appears in your list of enabled apps.

Note: - Ensure that your Nextcloud server is compatible with the version of the End-to-End Encryption app you intend to install. Compatibility details are usually listed on the app's page within the Apps management interface.

2. **Verify E2EE App Activation**: - Confirm the activation of the End-to-End Encryption app by navigating to 'Settings' under your profile menu. Check the 'Security' section to see if the End-to-End Encryption settings are available and configured correctly.



Contacts Interaction	1.8.0	✓ Featured	Disable
Dashboard	7.7.0	✓ Featured	Disable
Default encryption module	2.15.0	✓ Featured	Disable
Deleted files	1.17.0	✓ Featured	Disable
End-to-End Encryption	1.13.1	✓ Featured	Disable
Federation	1.17.0	✓ Featured	Disable
File reminders	1.0.0	✓ Featured	Disable

3. **Download and Install the Nextcloud Client**: This section outlines two methods to install the Nextcloud client on Linux: using Snap and using the APT repository. Follow the method that best suits your environment.

   Using Snap:

   - Update your package index and install snapd if it's not already installed:

```
sudo apt update
sudo apt install snapd
```

   - Install the Nextcloud desktop client using Snap:

```
sudo snap install nextcloud-desktop-client
```

   Using APT Repository (For Ubuntu 17.10 and later):

   - Open a terminal window and add the Nextcloud PPA to your system:

```
sudo add-apt-repository ppa:nextcloud-devs/client
```

   - Update the package index after adding the PPA:

```
sudo apt update
```

- Install the Nextcloud client:

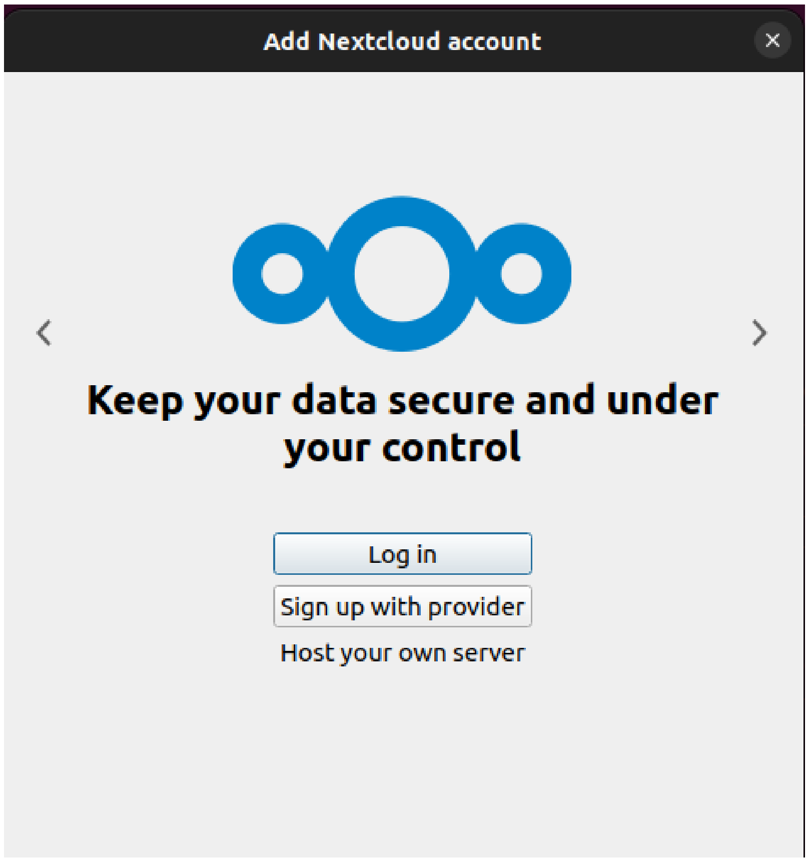
```
sudo apt install nextcloud-client
```

Note:

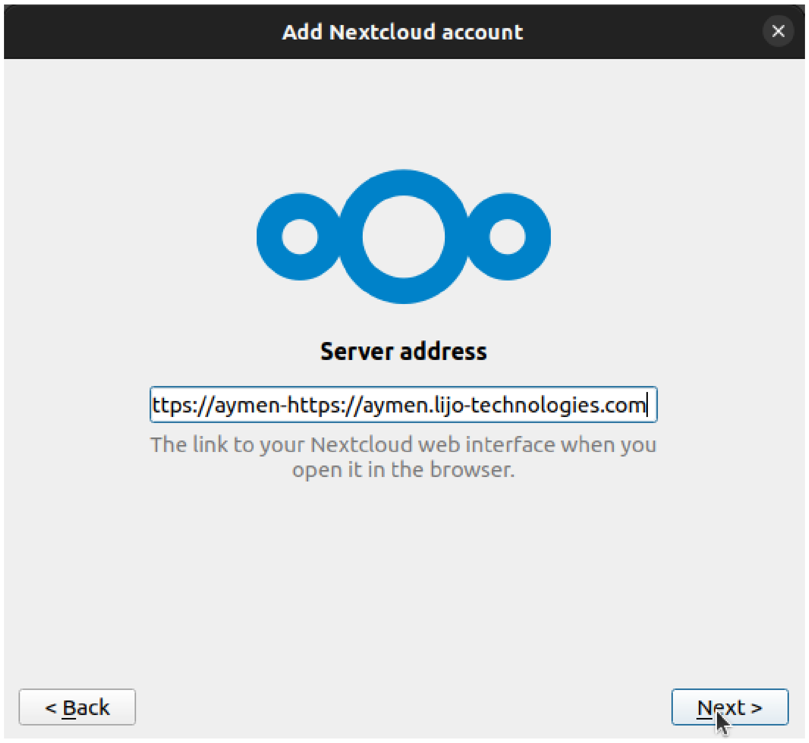
- The Snap method is generally simpler and provides automatic updates to the Nextcloud client.
- The APT method allows for more control over the installation process and may provide a more integrated system experience depending on your specific Linux distribution.

Choose the installation method that aligns with your system management preferences and follow the corresponding steps to complete the installation.

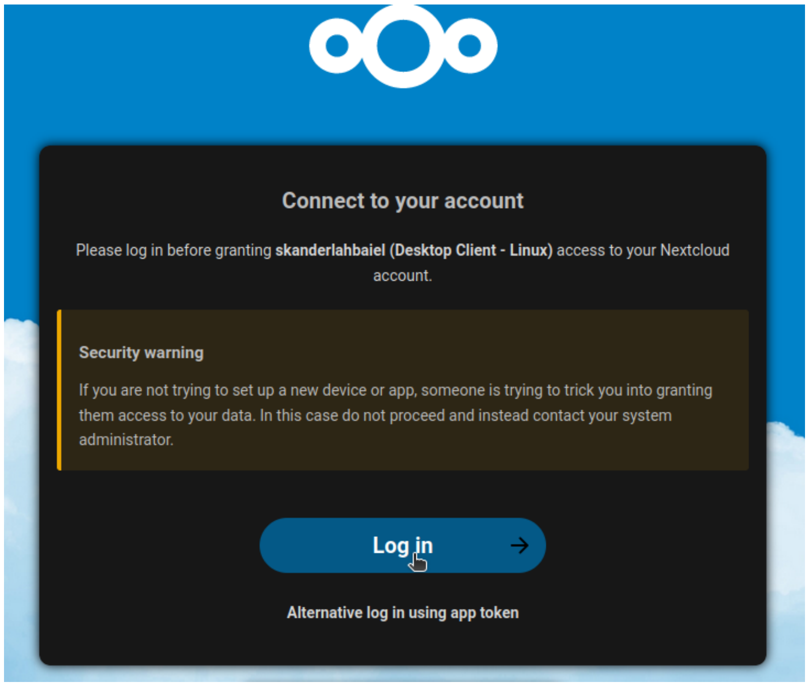
4. **Connect the Client to the Nextcloud Server**:
   - Launch the Nextcloud client.



   - Click on the loin button
   - Enter the URL of your Nextcloud server and provide the necessary credentials to establish a connection.


Add Nextcloud account
Server address
ttps://aymen-https://aymen.lijo-technologies.com
The link to your Nextcloud web interface when you open it in the browser.
< Back
Next >

- Accept the connection request from the server and log in to your Nextcloud account.

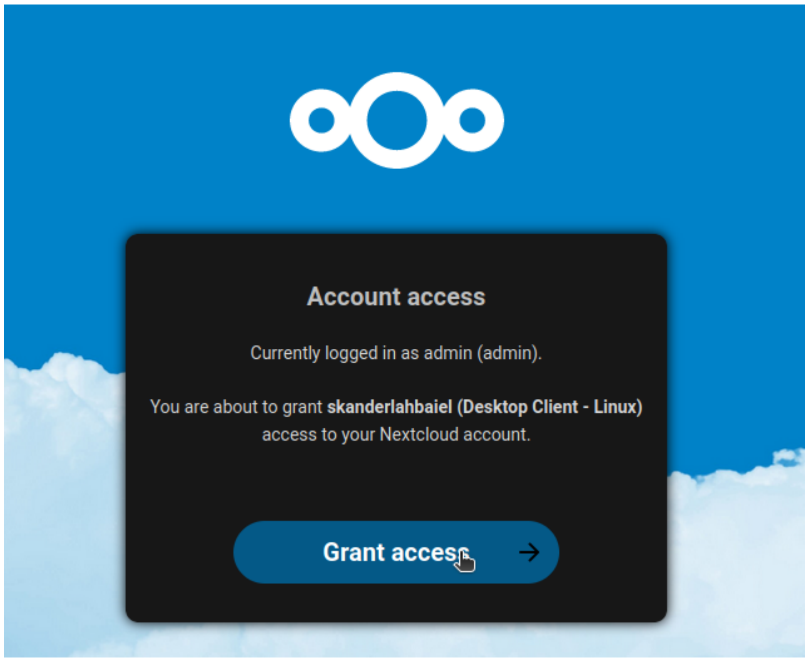


Connect to your account

Please log in before granting **skanderlahbaiel (Desktop Client - Linux)** access to your Nextcloud account.

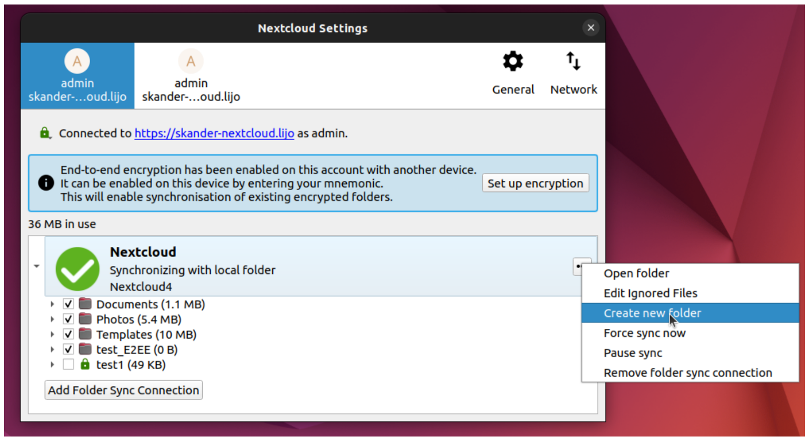
Security warning

If you are not trying to set up a new device or app, someone is trying to trick you into granting them access to your data. In this case do not proceed and instead contact your system administrator.

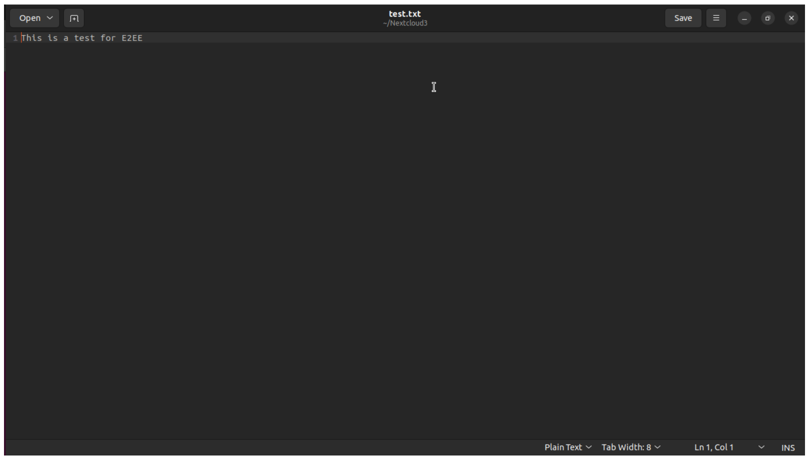

Account access
Currently logged in as admin (admin).
You are about to grant skanderlahbaiel (Desktop Client - Linux) access to your Nextcloud account.
Grant access

5. **Create and Encrypt a Folder**:

   - Within the Nextcloud client, create a new folder. You can name it *TestE2EE*.


Nextcloud Settings
admin
skander-...oud.lijo
admin
skander-...oud.lijo
General
Network
Connected to https://skander-nextcloud.lijo as admin.
End-to-end encryption has been enabled on this account with another device.
It can be enabled on this device by entering your mnemonic.
This will enable synchronisation of existing encrypted folders.
Set up encryption
36 MB in use
Nextcloud
Synchronizing with local folder
Nextcloud4
Documents (1.1 MB)
Photos (5.4 MB)
Templates (10 MB)
test_E2EE (0 B)
test1 (49 KB)
Add Folder Sync Connection
Open folder
Edit Ignored Files
Create new folder
Force sync now
Pause sync
Remove folder sync connection

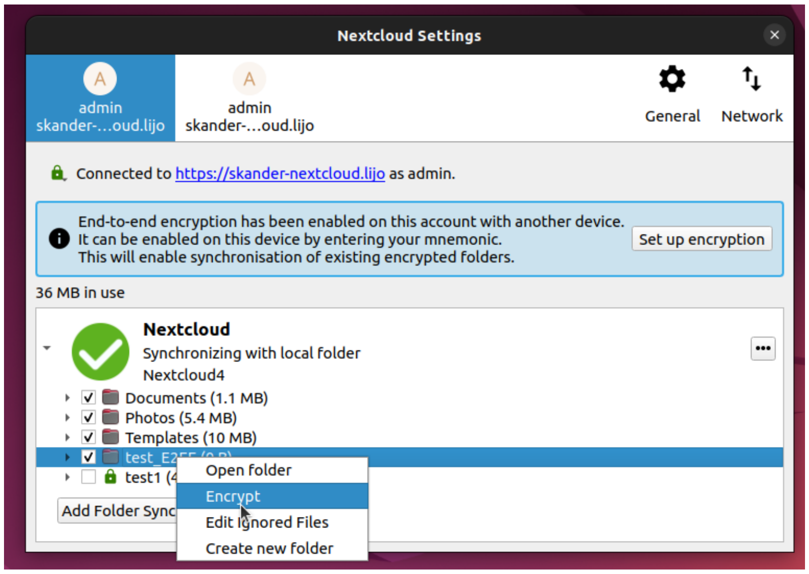
   - Inside this folder, create a new text file named *test.txt*.
   - Enter some content into the text file, for example, "This is a test for E2EE."



test.txt
~/Nextcloud3

```
This is a test for E2EE
```

- Save the file. Then, right-click on the *TestE2EE* folder and select the option to encrypt the folder.



Connected to https://skander-nextcloud.lijo as admin.

End-to-end encryption has been enabled on this account with another device.
It can be enabled on this device by entering your mnemonic.
This will enable synchronisation of existing encrypted folders.

36 MB in use

Nextcloud
Synchronizing with local folder
Nextcloud4

- ✓ Documents (1.1 MB)
- ✓ Photos (5.4 MB)
- ✓ Templates (10 MB)
- ✓ test_E2EE (0 B)
- test1 (4

6. **Verify the File on the Server**:

   - Access your Kubernetes cluster where Nextcloud is deployed.
   - Use the command line to run:

   ```
   kubectl get pv
   ```

   - Identify the persistent volume (PV) used by Nextcloud, usually tagged as *nextcloud-data*.
   - Describe the PV to find the storage path on the node:

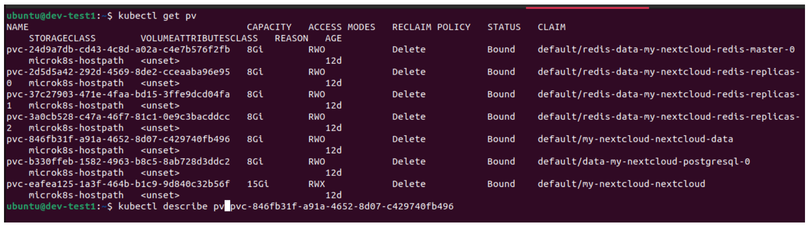
   ```
   kubectl describe pv nextcloud-data
   ```

   - Execute sudo -i and then access the server's storage path where Nextcloud data is stored:

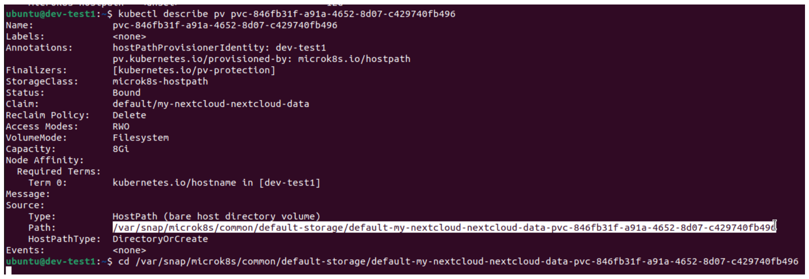
   ```
   sudo -i
   cd <path-to-my-nextcloud-data-pvc-directory>/data/admin/files
   cd test_E2EE
   ls
   ```

- Navigate to the directory containing user files and attempt to read the contents of *test.txt*:

```
cat <username>/files/<encrypted file name>.txt
```



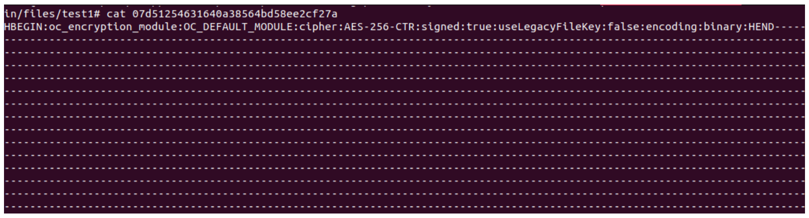
```
ubuntu@dev-test1:~$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
    STORAGECLASS        VOLUMEATTRIBUTESCLASS   REASON   AGE
pvc-24d9a7db-cd43-4c8d-a02a-c4e7b576f2fb   8Gi        RWO            Delete           Bound    default/redis-data-my-nextcloud-redis-master-0
    microk8s-hostpath   <unset>                          12d
pvc-2d5d5a42-292d-4569-8de2-cceaaba96e95   8Gi        RWO            Delete           Bound    default/redis-data-my-nextcloud-redis-replicas-
0   microk8s-hostpath   <unset>                          12d
pvc-37c27903-471e-4faa-bd15-3ffe9dcd04fa   8Gi        RWO            Delete           Bound    default/redis-data-my-nextcloud-redis-replicas-
1   microk8s-hostpath   <unset>                          12d
pvc-3a0cb528-c47a-46f7-81c1-0e9c3bacddcc   8Gi        RWO            Delete           Bound    default/redis-data-my-nextcloud-redis-replicas-
2   microk8s-hostpath   <unset>                          12d
pvc-846fb31f-a91a-4652-8d07-c429740fb496   8Gi        RWO            Delete           Bound    default/my-nextcloud-nextcloud-data
    microk8s-hostpath   <unset>                          12d
pvc-b330ffeb-1582-4963-b8c5-8ab728d3ddc2   8Gi        RWO            Delete           Bound    default/data-my-nextcloud-postgresql-0
    microk8s-hostpath   <unset>                          12d
pvc-eafea125-1a3f-464b-b1c9-9d840c32b56f   15Gi       RWX            Delete           Bound    default/my-nextcloud-nextcloud
    microk8s-hostpath   <unset>                          12d
ubuntu@dev-test1:~$ kubectl describe pv pvc-846fb31f-a91a-4652-8d07-c429740fb496
```



```
ubuntu@dev-test1:~$ kubectl describe pv pvc-846fb31f-a91a-4652-8d07-c429740fb496
Name:            pvc-846fb31f-a91a-4652-8d07-c429740fb496
Labels:          <none>
Annotations:     hostPathProvisionerIdentity: dev-test1
                 pv.kubernetes.io/provisioned-by: microk8s.io/hostpath
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:    microk8s-hostpath
Status:          Bound
Claim:           default/my-nextcloud-nextcloud-data
Reclaim Policy:  Delete
Access Modes:    RWO
VolumeMode:      Filesystem
Capacity:        8Gi
Node Affinity:
  Required Terms:
    Term 0:      kubernetes.io/hostname in [dev-test1]
Message:
Source:
    Type:          HostPath (bare host directory volume)
    Path:          /var/snap/microk8s/common/default-storage/default-my-nextcloud-nextcloud-data-pvc-846fb31f-a91a-4652-8d07-c429740fb496
    HostPathType:  DirectoryOrCreate
Events:          <none>
ubuntu@dev-test1:~$ cd /var/snap/microk8s/common/default-storage/default-my-nextcloud-nextcloud-data-pvc-846fb31f-a91a-4652-8d07-c429740fb496
```

7. **Check Encryption**:

   - Review the output from the *cat* command.
   - The contents of *test.txt* should appear garbled or encrypted, indicating that E2EE is functioning correctly.



```
in/files/test1# cat 07d51254631640a38564bd58ee2cf27a
HBEGIN:oc_encryption_module:OC_DEFAULT_MODULE:cipher:AES-256-CTR:signed:true:useLegacyFileKey:false:encoding:binary:HEND----
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
```

Conclusion

This manual test confirms whether E2EE is effectively encrypting files as expected, preventing unauthorized access even if server storage is compromised. Due to the nature of manual testing, this process is recommended to be conducted periodically to ensure ongoing compliance and functionality of the E2EE feature in Nextcloud.

Note

For automated tests or more complex scenarios, consider scripting interactions with the Nextcloud client using tools capable of GUI automation, tailored for your specific testing environment and requirements.

1.4.3 tests.test_create_user module

`tests.test_create_user.`**`add_user`**(*driver*, *username*, *display_name*, *password*, *email*, *group*, *quota*, *manager*)

Adds a new user to the application with specified attributes.

This function utilizes the *add_user_and_check_presence* utility function from the *functional_testing.utils.user_management* module to create a new user in the application. It then verifies the presence of the newly added user to ensure successful creation. The process involves automating browser interactions using Selenium WebDriver to fill and submit the new user form with provided user details.

Parameters

- **`driver`** (*`WebDriver`*) – An instance of Selenium WebDriver, used for automating interactions with the web application's interface.
- **`username`** (*`str`*) – The username for the new user. Must be unique within the application.
- **`display_name`** (*`str`*) – The display name of the new user. This is the name that will be displayed within the application's UI.
- **`password`** (*`str`*) – The password for the new user's account. Ensure compliance with application's password policy.
- **`email`** (*`str`*) – The email address associated with the new user. Must be in a valid email format.
- **`group`** (*`str, optional`*) – The group to which the new user will be assigned. Defaults to 'admin'. Available groups may vary based on application configuration.
- **`quota`** (*`str, optional`*) – The storage quota assigned to the new user's account. Defaults to '1 GB'. The format and available options may depend on application settings.
- **`manager`** (*`str, optional`*) – The username of the user's manager. Defaults to 'admin'. This is relevant in applications with hierarchical user management.

Returns

True if the new user is successfully added and their presence is verified within the application, False if the addition fails at any step or if the user cannot be verified post-creation.

Return type

bool

Raises

`Exception` – Propagates exceptions that may arise during the user creation or verification process, including issues with browser automation or failures in interacting with the application's UI.

Example:

```
>>> from selenium import webdriver
>>> driver = webdriver.Chrome()
>>> result = add_user(driver, "new_user", "New User", "securepassword", "new_
↪user@example.com",
...                     "users", "2 GB", "senior_manager")
>>> print(result)
True
```

Note:

This function requires the *NEW_USER_BUTTONS_XPATH* dictionary from the *func-*

tional_testing.config.configuration module, which contains XPaths for various UI elements involved in the user creation process.

1.4.4 tests.test_delete_user module

`tests.test_delete_user.`**`delete_user`**(*driver*, *username*)

Deletes a specified user from the application and verifies their absence.

This function navigates to the user management page, performs the deletion of a user based on the given username, and checks to ensure the user is no longer present in the application. It utilizes utility functions for navigation, user deletion, and verification of user absence. If the function encounters any exceptions during the process, it returns False, indicating the user was not successfully deleted.

Parameters

- **`driver`** (*`WebDriver`*) – An instance of Selenium WebDriver, used to automate browser interactions.
- **`username`** (*`str`*) – The username of the user to be deleted. This is used to identify the specific user in the user management interface.

Returns

Returns True if the user is successfully deleted and confirmed to be absent from the application. Returns False if the deletion process fails at any step or if the user is still present after the deletion attempt.

Return type

bool

Raises

`Exception` – Propagates any exceptions encountered during navigation, user deletion, or verification of user absence. It logs the exception before re-raising, providing context for the error.

Note:

- The function assumes that it's being called within the context of an authenticated session where the driver has access to the application's user management interface.
- This function relies on external utility functions *navigate_to_users_page*, *perform_delete_user*, and *check_user_absence* for various steps of the process. Ensure these are correctly implemented and accessible in the project.

Example:

```
>>> from selenium import webdriver
>>> driver = webdriver.Chrome()
>>> username = "testuser"
>>> deletion_success = delete_user(driver, username)
>>> print(deletion_success)
True
```

1.4.5 tests.test_file_delete module

`tests.test_file_delete.`**`delete_file`**(*driver*, *file_name*)

Deletes a specified file from NextCloud.

This function navigates to the NextCloud files section by ensuring the session is logged in. It then attempts to delete a file with the given name. Successful deletion is verified internally by the function *perform_delete_file*. If the deletion process encounters any issues, an exception is raised.

Parameters

- **`driver`** (*`WebDriver`*) – An instance of Selenium WebDriver used for automating interactions with the NextCloud web application. It should be properly initialized and configured prior to calling this function.
- **`file_name`** (*`str`*) – The name of the file to be deleted from NextCloud. The name should match exactly with the file present in NextCloud, including file extension.

Returns

True if the file is successfully deleted, False otherwise. This adjustment from the original docstring without a return statement provides clarity on the function's success through a boolean.

Return type

bool

Raises

`AssertionError` – If the file was not deleted successfully. This assertion could be raised within the *perform_delete_file* function based on the verification of the file's absence after the deletion attempt. Handling or logging this error in the calling context is recommended to make informed workflow decisions based on the file deletion success.

Example:

```
>>> from selenium import webdriver
>>> driver = webdriver.Chrome()
>>> file_name = "example_document.pdf"
>>> deletion_success = delete_file(driver, file_name)
>>> if deletion_success:
...     print(f"File {file_name} was successfully deleted.")
... else:
...     print(f"Failed to delete the file {file_name}.")
```

1.4.6 tests.test_file_download module

`tests.test_file_download.`**`download_file`**(*driver*, *file_name*)

Downloads a specified file from NextCloud.

This function first ensures that the session is logged into NextCloud, and that the file management screen is open, by invoking *ensure_logged_in_and_goto_files*. It then attempts to download a file with the specified *file_name*. The success of the download operation is determined by checking the presence of the downloaded file in a predetermined download directory. This function assumes that *perform_download_file* is responsible for both downloading the file and verifying its presence in the *DOWNLOAD_DIRECTORY*.

Parameters

- **`driver`** (*`WebDriver`*) – An instance of Selenium WebDriver, used for automating interactions with the web application. The driver should be configured to handle file downloads, including specifying the download directory.

- **file_name** (*str*) – The name of the file to be downloaded from NextCloud. The file name should include any file extension if applicable and match exactly with a file available in NextCloud.

Returns
True if the file is successfully downloaded and verified to be present in the local download directory, False otherwise.

Return type
bool

Raises
Exception – If any error occurs during the process of navigating to the files page, initiating the file download, or verifying the file's presence in the local directory.

Note:

- The function leverages configuration variables from *functional_testing.config.configuration* for navigation and login.
- The download directory is set via the *DOWNLOAD_DIRECTORY* configuration variable. This directory should be correctly configured in both the WebDriver setup and the environment to ensure downloaded files are saved and found as expected.
- The function prints a success message in green upon successful download. This visual cue can be helpful in interactive or manual testing scenarios.

```python
from selenium import webdriver
driver = webdriver.Chrome()
file_name = "sample_document.pdf"
success = download_file(driver, file_name)
if success:
    print(f"The file {file_name} was successfully downloaded.")
else:
    print(f"Failed to download the file {file_name}.")
```

1.4.7 tests.test_file_integrity module

tests.test_file_integrity.**file_integrity**(*file_name*)

Verifies the integrity of a file by comparing the hash of the uploaded file with that of the downloaded file.

This function aims to ensure data integrity by verifying that a file, once uploaded and then downloaded from NextCloud, remains unaltered. It computes and compares the hash values of both the uploaded and downloaded versions of the file located in predetermined directories. A match in hash values indicates that the file has maintained its integrity throughout the upload and download process.

Parameters
file_name (*str*) – The name of the file for which integrity is being checked. This name should include any file extension and match exactly between the uploaded and downloaded files.

Returns
Returns True if the hash values of the uploaded and downloaded files match, indicating the files are identical and integrity is preserved. Returns False if there is a mismatch in hash values, suggesting the files are different and integrity may have been compromised.

Return type
bool

Raises
`AssertionError` – Raises an assertion error if the hash comparison fails, indicating the uploaded and downloaded files are not the same. This may point to issues in the file transfer process or unauthorized alterations of the file content.

Example:

```
>>> file_name = "example_document.pdf"
>>> integrity_check = file_integrity(file_name)
>>> if integrity_check:
...     print("File integrity verified.")
... else:
...     print("File integrity compromised.")
```

Note: The function relies on the *compare_hashes* utility to perform the hash computation and comparison. Ensure that *DOWNLOAD_DIRECTORY* and *CREATED_FILES_PATH* in the configuration module accurately reflect the paths to where the NextCloud application stores downloaded files and where uploaded files are kept, respectively. It assumes that the upload and the download have been performed successfully.

1.4.8 tests.test_file_share module

`tests.test_file_share.`**`locate_and_click_share_button`**(*driver*, *file_name*)

Shares a specified file internally and externally within NextCloud.

This function handles multiple steps to share a file stored in NextCloud. Initially, it navigates to the files management page, locates the desired file, and then initiates the sharing process. It performs both internal sharing with a specified recipient and generates external sharing links. The function also validates the accessibility of these shared links by simulating external access to ensure that the sharing functionality works as expected.

Args:

driver (WebDriver): An instance of Selenium WebDriver used for automating browser interactions. It must be
initialized and configured before being passed to this function.

username (str): The username of the recipient with whom the file will be shared internally. This user must
exist within the NextCloud system.

password (str): The password for the above username, used to validate internal sharing functionality by
attempting to access the shared file.

file_name (str): The name of the file to be shared. The file should already exist in the NextCloud directory
that the initiating user has access to.

Returns:

bool: True if both internal and external sharing links are generated and validated successfully, indicating
that the file has been shared properly. Returns False if any step in the sharing or validation process fails.

Raises:

Exception: If an error occurs at any point during the file sharing process, including issues with locating
the file, sharing operations, link generation, or access validations. The specific error message is printed to the console.

Example:

```
>>> from selenium import webdriver
>>> driver = webdriver.Chrome()
>>> username = "example_user"
>>> password = "password123"
>>> file_name = "document.pdf"
>>> share_success = share_file(driver, username, password, file_name)
>>> if share_success:
...     print("File shared successfully.")
... else:
...     print("File sharing failed.")
```

Note:

- This function assumes that the user initiating the share (typically an admin or the file owner) is logged into NextCloud at the start of the function.
- The function performs a logout operation as part of its external link validation step, and then logs back in as the admin user. Ensure that any necessary session data is saved or that subsequent operations account for this logout/login cycle.

`tests.test_file_share.`**`share_file`**(*driver*, *username*, *password*, *file_name*, *LOGIN_BUTTON_XPATH="//button[@class='button-vue button-vue--icon-and-text button-vue--vue-primary button-vue--wide']"*)

Shares a specified file internally and externally within NextCloud.

This comprehensive function handles multiple steps to share a file stored in NextCloud. Initially, it navigates to the files management page, locates the desired file, and then initiates the sharing process. It performs both internal sharing with a specified recipient and generates external sharing links. The function also validates the accessibility of these shared links by simulating external access to ensure that the sharing functionality works as expected.

Parameters

- **driver** (*WebDriver*) – An instance of Selenium WebDriver used for automating browser interactions. It must be initialized and configured before being passed to this function.
- **username** (*str*) – The username of the recipient with whom the file will be shared internally. This user must exist within the NextCloud system.
- **password** (*str*) – The password for the above username, used to validate internal sharing functionality by attempting to access the shared file.
- **file_name** (*str*) – The name of the file to be shared. The file should already exist in the NextCloud directory that the initiating user has access to.

Returns

True if both internal and external sharing links are generated and validated successfully, indicating that the file has been shared properly. Returns False if any step in the sharing or validation process fails.

Return type

bool

Raises

Exception – If an error occurs at any point during the file sharing process, including issues with locating the file, sharing operations, link generation, or access validations. The specific error message is printed to provide insights into the encountered issue.

Example:

```
>>> from selenium import webdriver
>>> driver = webdriver.Chrome()
>>> username = "example_user"
>>> password = "password123"
>>> file_name = "document.pdf"
>>> share_success = share_file(driver, username, password, file_name)
>>> if share_success:
...     print("File shared successfully.")
... else:
...     print("File sharing failed.")
```

Note:

- This function assumes that the user initiating the share (typically an admin or the file owner) is logged into NextCloud at the start of the function.
- The function performs a logout operation as part of its external link validation step, and then logs back in as the admin user. Ensure that any necessary session data is saved or that subsequent operations account for this logout/login cycle.

1.4.9 tests.test_file_upload module

`tests.test_file_upload.`**`upload_file`**(*driver*, *file_path*)

Uploads a file to NextCloud and verifies its presence in the file list in Nextcloud UI.

This function automates the process of uploading a file to NextCloud using Selenium WebDriver. It ensures that the user is logged in and navigates to the files section. The function then initiates the file upload process and verifies whether the uploaded file appears in NextCloud's file list. If the file is not found within a specified timeout, the upload is considered unsuccessful.

Parameters

- **driver** (*WebDriver*) – The Selenium WebDriver instance used to automate browser interactions. It should be initialized and configured to the target NextCloud instance prior to calling this function.
- **file_path** (*str*) – The absolute path of the file on the local system that is to be uploaded. Ensure that the path is accessible and readable by the script to avoid errors during the upload process.

Returns

True if the file is successfully uploaded and verified to be present in NextCloud's file list. False if the file does not appear in the list after the upload process, indicating a failure.

Return type

bool

Raises

Exception – General exception if an error occurs at any point during the login, navigation, upload process, or file presence check. The specific exception message is printed to provide insights into the failure reason.

Note:

- This function assumes that *TARGET_URL*, *USERNAME*, *PASSWORD*, *LOGIN_BUTTON_XPATH*, *FILES_TAB_XPATH*, *NEW_FILE_OR_FOLDER_MENU_XPATH*, and *FILE_UPLOAD_START_ID* are correctly configured in the *functional_testing.config.configuration* module.
- Ensure that the WebDriver instance (*driver*) has been correctly authenticated with NextCloud prior to calling this function if not relying on the function's login mechanism. It's recommended to use the login function before calling this function.

```
from selenium import webdriver
driver = webdriver.Chrome()
file_path = "/path/to/your/file.txt"
upload_success = upload_file(driver, file_path)
if upload_success:
    print("File upload successful.")
else:
    print("File upload failed.")
```

1.4.10 tests.test_login module

`tests.test_login.`**`login`**(*driver*)

Authenticates a user into NextCloud using predefined credentials.

This function orchestrates the process of logging into a NextCloud instance. It begins by navigating to the NextCloud login page, specified by *TARGET_URL*, and then proceeds to input the predefined credentials (*USERNAME* and *PASSWORD*) into the login form. The function checks for successful authentication by verifying the presence of elements unique to authenticated sessions. If the login attempt faces any issues, the function gracefully handles exceptions and provides feedback.

Parameters

driver (*WebDriver*) – An instance of a Selenium WebDriver, used to automate web browser interaction. The driver must be initialized and configured prior to calling this function.

Returns

Indicates the outcome of the login attempt. Returns True if the login process completes successfully, indicating that the user has been authenticated. Returns False if there are any issues during the login process, suggesting that the authentication did not occur.

Return type

bool

Raises

Exception – Raises a generic exception if any unexpected errors occur during the execution of the login process. The exception captures and suppresses detailed error information to prevent potential sensitive data exposure, adhering to security best practices.

Example

```
>>> from selenium import webdriver
>>> driver = webdriver.Chrome()
```

(continues on next page)

(continued from previous page)

```
>>> login_success = login(driver)
>>> if login_success:
...     print("Login successful.")
... else:
...     print("Login failed.")
```

Note

The login process relies on several configuration variables defined in the *functional_testing.config.configuration* module, including *TARGET_URL*, *USERNAME*, *PASSWORD*, and *LOGIN_BUTTON_XPATH*. Ensure these variables are accurately set to reflect the current state of the NextCloud login page and user credentials.

1.4.11 tests.test_logout module

`tests.test_logout.`**`logout`**`(`*driver*`)`

Performs logout from the application using the provided WebDriver instance.

This function navigates the application's UI to log out the currently authenticated user. It specifically handles the UI interactions required to open the settings menu, click the logout button, and verify the logout process by checking for the presence of the login button. It uses a series of utility functions tailored to interact with specific UI elements defined in the configuration module.

Parameters

`driver` (*`WebDriver`*) – An instance of Selenium WebDriver. This is used to automate the interaction with the web browser, enabling the function to perform actions like clicking and checking the presence of UI elements.

Returns

Indicates the outcome of the logout attempt. Returns True if the logout process is completed successfully, evidenced by the appearance of the login button. Returns False if the logout process fails at any step, either due to UI elements not being found or other exceptions.

Return type

bool

Raises

- **`NoSuchElementException`** – Raised if any expected UI element involved in the logout process is not found within the current page. This exception points to possible changes in the UI or an incorrect navigation state.
- **`Exception`** – A generic exception is raised for any other errors encountered during the logout process. The function catches and logs these exceptions, providing a message detailing the encountered issue.

```
Example:
    driver = webdriver.Chrome()
    logout_success = logout(driver)
    if logout_success:
        print("Successfully logged out.")
    else:
        print("Logout failed.")
```

Note: The function assumes that the user is already logged in and that the WebDriver instance (*driver*) is in a state that allows direct interaction with the application's logout functionality. It relies on accurate XPath locators for the settings menu and logout button, specified in *OPEN_SETTINGS_MENU_XPATH*, *CLOSED_SETTINGS_MENU_XPATH*, and *LOGOUT_BUTTON_PATH*, which should be configured in the *functional_testing.config.configuration* module before invoking this function.

1.4.12 tests.test_modify_quota module

`tests.test_modify_quota.`**`modify_user_quota`**`(`*driver*, *username*, *quota*, *password*`)`

modiy_user_quota _summary_

_extended_summary_

Parameters

`driver` (*_type_*) – _description_

1.4.13 Module contents

1.5 utils package

1.5.1 Submodules

1.5.2 utils.authentication_management module

`utils.authentication_management.`**`check_if_loggedIn`**`(`*driver*, *wait_time*`)`

Checks if the user is logged in by verifying the presence of a token in the cookies.

Args:

driver: The Selenium WebDriver instance. wait_time: The maximum time to wait for the user to be logged in, in seconds.

Returns:

True if the user is logged in, False otherwise.

`utils.authentication_management.`**`check_logout`**`(`*driver*, *LOGIN_BUTTON_XPATH="//button[@class='button-vue button-vue--icon-and-text button-vue--vue-primary button-vue--wide']"*`)`

Checks if the user has successfully logged out by waiting for the login button to appear.

Args:

driver: The WebDriver instance used for interacting with the browser. LOGIN_BUTTON_XPATH: The XPath of the login button element.

Returns:

True if the logout was successful, False otherwise.

`utils.authentication_management.`**`check_successful_page_load`**`(`*driver*, *wait_time*`)`

Checks if the page load is successful by verifying the presence of specific elements on the page.

Args:

driver (WebDriver): The WebDriver instance used for interacting with the browser. wait_time (int): The maximum time to wait for the elements to load, in seconds.

Returns:

str: The ID of the first element that successfully loads within the specified wait time.
If none of the elements load within the wait time, returns None.

`utils.authentication_management.`**`click_on_logout`**`(`*driver*, *LOGOUT_BUTTON_PATH*`)`

Clicks on the logout button.

Args:
driver (WebDriver): The WebDriver instance used to interact with the browser. LOGOUT_BUTTON_PATH (str): The XPath of the logout button element.

Returns:
None

`utils.authentication_management.`**`get_token_from_cookies`**`(`*driver*`)`

Retrieves the 'nc_token' from the browser cookies.

Args:
driver (WebDriver): The web driver instance controlling the browser.

Returns:
str: The value of 'nc_token' if found, None otherwise.

`utils.authentication_management.`**`green_text`**`(`*text*`)`

`utils.authentication_management.`**`load_nextcloud_page`**`(`*driver*, *TARGET_URL*`)`

Loads the NextCloud page.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. TARGET_URL (str): The URL of the NextCloud instance.

Returns:
None

`utils.authentication_management.`**`login_to_nextcloud`**`(`*driver*, *TARGET_URL*, *USERNAME*, *PASSWORD*, *LOGIN_BUTTON_XPATH*`)`

Fills the login form, submits and checks for successful login.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. TARGET_URL (str): The URL of the NextCloud instance. USERNAME (str): The username to use for logging in. PASSWORD (str): The password to use for logging in. LOGIN_BUTTON_XPATH (str): The XPath expression for the login button.

Returns:
bool: True if the login is successful, False otherwise.

`utils.authentication_management.`**`orange_text`**`(`*text*`)`

`utils.authentication_management.`**`perform_logout`**`(`*driver*, *LOGOUT_BUTTON_PATH*, *LOGIN_BUTTON_XPATH*, *OPEN_SETTINGS_MENU_XPATH*, *CLOSED_SETTINGS_MENU_XPATH*`)`

Logs out of NextCloud.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. LOGOUT_BUTTON_PATH (str): The XPath of the logout button element. LOGIN_BUTTON_XPATH (str): The XPath of the login button element.

Returns:
bool: True if the logout is successful, False otherwise.

`utils.authentication_management.`**`toggle_settings_menu`**`(`*driver*, *OPEN_SETTINGS_MENU_XPATH*, *CLOSED_SETTINGS_MENU_XPATH*, *expected_state='open'*`)`

Toggles the settings menu to the expected state ('open' or 'closed').

Args:
driver (WebDriver): The WebDriver instance used for browser automation. expected_state (str): The expected state of the settings menu. Defaults to 'open'.

1.5.3 utils.file_creation module

1.5.4 utils.file_management module

`utils.file_management.`**`calculate_sha256`**`(`*file_path*`)`

Calculate the SHA-256 hash of a file.

Args:
file_path (str): The path to the file.

Returns:
str: The SHA-256 hash of the file.

`utils.file_management.`**`check_external_file_sharing`**`(`*driver*, *file_name*`)`

Checks whether the page contains the file name and the download button.

`utils.file_management.`**`check_file_deletion`**`(`*driver*, *file_element*, *wait_time*`)`

Checks if the uploaded file is absent in NextCloud.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. file_element (WebElement): The WebElement instance of the file. wait_time (int): The maximum time (in seconds) to wait for the file to be absent.

Returns:
True if the file is absent within the specified wait time, False otherwise.

`utils.file_management.`**`check_file_presence`**`(`*driver*, *file_xpath*, *wait_time*`)`

Checks if the uploaded file is present in NextCloud.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. file_xpath (str): The XPath expression to locate the file element. wait_time (int): The maximum time (in seconds) to wait for the file to be present.

Returns:
True if the file is present within the specified wait time, False otherwise.

`utils.file_management.`**`check_files_container_presence`**`(`*driver*, *timeout=10*`)`

Checks if the specific element is present on the page.

Args:
driver (WebDriver): The Selenium WebDriver instance. timeout (int): Maximum time to wait for the element to be present.

Returns:
bool: True if the element is found within the timeout, False otherwise.

`utils.file_management.`**`check_internal_file_share`**(*driver*, *file_name*, *internal_share_link*, *username*, *password*, *LOGIN_BUTTON_XPATH*)

`utils.file_management.`**`check_internal_sharing`**(*driver*, *file_name*)

`utils.file_management.`**`check_url_login_page`**(*driver*)

Checks if the current URL is the login page.

`utils.file_management.`**`click_new_fileOrFolder`**(*driver*, *NEW_FILE_OR_FOLDER_MENU_XPATH*)

`utils.file_management.`**`compare_hashes`**(*file_name*, *DOWNLOAD_DIRECTORY*, *CREATED_FILES_PATH*)

This function compares the SHA-256 hashes of the uploaded and downloaded files.

It calculates the SHA-256 hash of both the uploaded and downloaded files and compares them. If the hashes are not the same, it raises an AssertionError.

Args:
file_name (str): The name of the file. DOWNLOAD_DIRECTORY (str): The directory where the downloaded file is stored. CREATED_FILES_PATH (str): The directory where the uploaded file is stored.

Returns:
bool: True if the hashes are the same, False otherwise.

Raises:
Exception: If an error occurs while comparing the hashes.

`utils.file_management.`**`create_file`**(*file_size_mb*, *directory='./created_files'*, *base_name='test'*, *extension='.txt'*)

Creates a file of a specific size in megabytes in the specified directory with an automatically generated name.

Args: file_size_mb (int): The size of the file in megabytes. directory (str): The directory where the file will be created. base_name (str): Base name for the file. extension (str): File extension.

Returns: str: The path of the created file. str: The name of the created file.

`utils.file_management.`**`create_testing_files`**(*FILE_SIZE_MEDIUM*, *FILE_SIZE_LARGE*, *CREATED_FILES_PATH*, *TEST_FILE_BASE_MEDIUM*, *TEST_FILE_BASE_LARGE*, *EXTENSION*)

Creates a medium size file and a large size file for testing purposes.

Args:
FILE_SIZE_MEDIUM (int): The size of the medium size file in bytes. FILE_SIZE_LARGE (int): The size of the large size file in bytes. CREATED_FILES_PATH (str): The path to the directory where the files will be created. base_name (str): The base name of the files. extension (str): The extension of the files.

Returns:
str: The path to the medium size file. str: The name of the medium size file. str: The path to the large size file. str: The name of the large size file.

`utils.file_management.`**`ensure_element_interactable`**(*driver*, *element*)

Ensures the element is in view and interactable by scrolling into view and checking clickability.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. element (WebElement): The Selenium WebElement to make interactable.

`utils.file_management.`**`ensure_logged_in_and_goto_files`**`(`*driver*, *TARGET_URL*, *USERNAME*, *PASSWORD*, *LOGIN_BUTTON_XPATH*, *FILES_TAB_XPATH*`)`

This function checks if the the file container is present first, if it is then it passes to proceed to other tasks, else it logs in and navigates to the files page.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. TARGET_URL (str): The URL of the NextCloud instance. USERNAME (str): The username to use for logging in. PASSWORD (str): The password to use for logging in. LOGIN_BUTTON_XPATH (str): The XPath expression for the login button. FILES_TAB_XPATH (str): The XPath of the "Files" button. NEW_FILE_OR_FOLDER_MENU_XPATH (str): The XPath of the "New file/folder menu" button.

Returns:
None

`utils.file_management.`**`extract_file_type`**`(`*file_path*`)`

Extracts the file type from the given file path.

Args:
file_path (str): The path of the file.

Returns:
str: The file type.

Example:

```
>>> extract_file_type('/path/to/file.txt')
'txt'
```

`utils.file_management.`**`extract_filename`**`(`*file_path*`)`

Extracts the filename from a given file path.

Args:
file_path (str): The path of the file.

Returns:
str: The filename extracted from the file path.

`utils.file_management.`**`find_delete_button`**`(`*driver*, *file_name*`)`

Finds the delete button of the file in NexcCloud UI.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. file_name (str): The name of the file to find.

Returns:
Web element: A tuple containing the WebElement instance of the delete button.

`utils.file_management.`**`generate_external_share_verification_element`**`(`*driver*, *file_name*`)`

Generates the XPath for the verification element for external file sharing.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. file_name (str): The name of the file to generate the XPath for.

Returns:
str: The generated XPath for the verification element.

`utils.file_management.`**`generate_file_share_button`**(*driver*, *file_name*)

Locates and returns the share button for a given file in a web page.

Args:
driver (WebDriver): The web driver instance controlling the browser. file_name (str): The name of the file for which to find the share button.

Returns:
WebElement: The share button element for the specified file. Returns None if the share button could not be found.

`utils.file_management.`**`generate_internal_sharing_verification_element_exists`**(*file_name*)

Generates the XPath for the verification element for internal file sharing.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. file_name (str): The name of the file to generate the XPath for.

Returns:
str: The generated XPath for the verification element.

`utils.file_management.`**`get_external_share_link`**(*driver*, *EXTERNAL_LINK_COPY_BUTTON_XPATH*, *EXTERNAL_LINK_COPY_BUTTON_XPATH_HREF*)

Gets the external share link of a file.

Args:
driver: The WebDriver instance used to interact with the browser. EXTERNAL_LINK_COPY_BUTTON_XPATH: The XPath of the external link copy button.

Returns:
The external share link of the file.

Raises:
Exception: If an error occurs while getting the external share link.

`utils.file_management.`**`get_internal_share_link`**(*driver*, *INTERNAL_LINK_COPY_BUTTON_XPATH*)

Gets the internal share link of a file.

Args:
driver: The WebDriver instance used to interact with the browser. INTERNAL_LINK_COPY_BUTTON_XPATH: The XPath of the internal link copy button.

Returns:
The internal share link of the file.

Raises:
Exception: If an error occurs while getting the internal share link.

`utils.file_management.`**`goto_files`**(*driver*, *FILES_TAB_XPATH*)

Navigates to the files page and opens the file upload dialog.

Args:
driver (WebDriver): The Selenium WebDriver instance. FILES_TAB_XPATH (str): The XPath of the "Files" button. NEW_FILE_OR_FOLDER_MENU_XPATH (str): The XPath of the "New file/folder menu" button.

Raises:
Exception: If an error occurs during navigation.

Returns:
None

`utils.file_management.`**`ifLogin_delete_cookies`**(*driver*, *TARGET_URL='http://skander-nextcloud.lijo'*)

Checks whether the page contains the file name and the download button.

Args:
driver (WebDriver): The WebDriver instance used to interact with the browser. internal_share_link (str): The internal file sharing link to navigate to.

Returns:
bool: True if the page contains the file name and the download button, False otherwise.

`utils.file_management.`**`is_element_clickable`**(*driver*, *element*)

Checks if the web page element is clickable.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. element (WebElement): The Selenium WebElement to check for clickability.

Returns:
bool: True if the element is clickable, False otherwise.

`utils.file_management.`**`locate_file_in_nextcloud`**(*driver*, *file_name*)

Locates a file in NextCloud.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. file_name (str): The name of the file to locate.

Returns:
bool: True if the file is found, False otherwise.

`utils.file_management.`**`logout_and_goto_external_link`**(*driver*, *LOGOUT_BUTTON_PATH*, *LOGIN_BUTTON_XPATH*, *OPEN_SETTINGS_MENU_XPATH*, *CLOSED_SETTINGS_MENU_XPATH*, *EXTERNAL_LINK*)

`utils.file_management.`**`perform_delete_file`**(*driver*, *file_name*)

`utils.file_management.`**`perform_download_file`**(*driver*, *file_name*, *DOWNLOADS_PATH*, *max_wait_time=500*, *check_interval=5*)

Downloads a file from NextCloud.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. file_name (str): The name of the file to download. DOWNLOADS_PATH (str): The path where the downloaded file will be saved. max_wait_time (int, optional): The maximum time (in seconds) to wait for the file to be downloaded. Defaults to 500. check_interval (int, optional): The interval (in seconds) between each check for the downloaded file. Defaults to 5.

Raises:
Exception: If the maximum wait time is exceeded while waiting for the file to download.

Returns:
None

`utils.file_management.`**`perform_file_upload`**(*driver*, *file_path*, *file_upload_id*)

Uploads a file and returns the XPath for checking its presence.

Args:
driver (WebDriver): The WebDriver instance. file_path (str): The path of the file to upload. file_upload_id (str): The ID of the file upload input element.

Returns:
str: The XPath for checking the presence of the uploaded file.

`utils.file_management.`**`scroll_element_into_view`**`(`*driver*, *element*`)`

Scrolls the web page to bring the element into the viewport.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. element (WebElement): The Selenium WebElement to scroll into view.

`utils.file_management.`**`scroll_to_bottom_dynamic`**`(`*driver*, *timeout=60*`)`

`utils.file_management.`**`scroll_with_mouse_wheel`**`(`*driver*, *element*, *deltaY=100*`)`

Scroll within an element using the mouse wheel.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. element (WebElement): The element to scroll within. deltaY (int): The vertical scroll amount (positive scrolls down, negative scrolls up).

`utils.file_management.`**`share_file_with_recipient`**`(`*driver*, *username*, *file_name*, *SHARE_WITH_ANOTHER_USER_INPUT_XPATH*, *SAVE_SHARING_BUTTON_XPATH*`)`

Share a file with a recipient.

Args:
driver: The WebDriver instance. username: The username of the recipient. file_name: The name of the file to be shared. SHARE_WITH_ANOTHER_USER_INPUT_XPATH: The XPath of the input field for sharing with another user. SAVE_SHARING_BUTTON_XPATH: The XPath of the button to save the sharing.

Raises:
Exception: If an error occurs while locating and filling the search field.

`utils.file_management.`**`toggle_file_upload`**`(`*driver*, *FILE_UPLOAD_START_ID*`)`

Toggles the file upload window by revealing the file upload input element.

Args:
driver: The WebDriver instance used to interact with the browser. FILE_UPLOAD_START_ID: The ID of the file upload input element.

Raises:
AssertionError: If the upload field is not visible.

Returns:
None

1.5.5 utils.user_management module

`utils.user_management.`**`add_user_and_check_presence`**`(`*driver*, *username*, *display_name*, *password*, *email*, *group*, *quota*, *manager*, *NEW_USER_BUTTONS_XPATH*`)`

Adds a user to the application and verifies their presence in the user list.

This function navigates the application's UI to add a new user with provided details. After adding, it attempts to locate the user in the application's user list to verify successful addition. If the user is found, the operation is considered successful.

Parameters

- **`driver`** (*`WebDriver`*) – WebDriver instance for interacting with the web application.
- **`username`** (*`str`*) – The username of the new user.
- **`display_name`** (*`str`*) – The display name of the new user.
- **`password`** (*`str`*) – The password for the new user.
- **`email`** (*`str`*) – The email address of the new user.
- **`group`** (*`str`*) – The group to which the new user belongs.
- **`quota`** (*`str`*) – The quota assigned to the new user.
- **`manager`** (*`str`*) – The manager of the new user.
- **`NEW_USER_BUTTONS_XPATH`** (*`dict`*) – A collection of keyword arguments mapping descriptive names to XPaths. These XPaths are used to interact with various UI elements necessary for user creation. Expected keys include:
  - OPEN_SETTINGS_MENU_XPATH: XPath to open the settings menu.
  - USERS_BUTTON_XPATH: XPath to access the users section.
  - NEW_USER_BUTTON: XPath to initiate new user creation.
  - NEW_USERNAME_INPUT: XPath for entering the user's username.
  - NEW_USER_ADD_BUTTON: XPath to finalize adding the new user.

Returns
True if the user is successfully added and found in the user list; returns False otherwise.

Return type
bool

Raises
`Exception` – Raises an exception if there's an issue during the user addition process or while checking the user's presence.

`utils.user_management.`**`check_presence_user_management_page`**`(`*driver*, *OPEN_SETTINGS_MENU_XPATH*, *CLOSED_SETTINGS_MENU_XPATH*, *USERS_BUTTON_XPATH*`)`

Checks if the user management page is open. Else, it opens the user management page.

`utils.user_management.`**`check_user_absence`**`(`*driver*, *username*`)`

Checks if the user is absent from the user list.

Args:
driver (WebDriver): The WebDriver instance used for interacting with the web page. username (str): The username of the user to check for absence.

Returns:
None

Raises:
AssertionError: If the user is not deleted or if any exception occurs during the process.

`utils.user_management.`**`check_user_presence`**`(`*driver*, *username*`)`

Checks if the user is present in the user list.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. username (str): The username of the user to check for.

Returns:
bool: True if the user is present, False otherwise.

`utils.user_management.`**`click_on_create_user`**`(`*driver*, *NEW_USER_ADD_BUTTON*`)`

Clicks on the "Create user" button.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. CREATE_USER_BUTTON_XPATH (str): The XPath of the "Create user" button.

Returns:
None

`utils.user_management.`**`click_users_button`**`(`*driver*, *USERS_BUTTON_XPATH*`)`

Clicks on the "Users" button.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. USERS_BUTTON_XPATH (str): The XPath of the "Users" button.

Returns:
None

`utils.user_management.`**`fill_user_form`**`(`*driver*, *username*, *display_name*, *password*, *email*, *group*, *quota*, *manager*, *NEW_USER_BUTTONS_XPATH*`)`

Fills the user form with provided details in a NextCloud application.

This function automates the process of filling out the user creation form in the NextCloud UI. It inputs the provided user details into the form fields, selects the specified user group, and sets the user quota. The function relies on a dictionary of XPaths to various form elements to interact with them.

Parameters

- **driver** (*WebDriver*) – The Selenium WebDriver instance used for browser automation.
- **username** (*str*) – Username to be entered in the form.
- **display_name** (*str*) – Display name to be entered in the form.
- **password** (*str*) – Password to be entered in the form.
- **email** (*str*) – Email address to be entered in the form.
- **group** (*str*) – User group to be selected from a dropdown.
- **quota** (*str*) – User data quota to be entered in the form.
- **manager** (*str*) – Manager's name to be entered in the form.
- **NEW_USER_BUTTONS_XPATH** (*dict*) – A dictionary mapping descriptive names to XPaths for interacting with the form.

Returns
Indicates whether the user form was successfully filled.

Return type
bool

Raises
: **`Exception`** – If an error occurs while filling the user form, providing the encountered error message.

Example:

```
>>> from selenium import webdriver
>>> driver = webdriver.Chrome()
>>> username = "newuser"
>>> display_name = "New User"
>>> password = "securepassword"
>>> email = "newuser@example.com"
>>> group = "Users"
>>> quota = "1GB"
>>> manager = "admin"
>>> NEW_USER_BUTTONS_XPATH = {
...     'NEW_USER_QUOTA_INPUT': 'xpath_here',
...     'NEW_USER_MANAGERS_INPUT': 'xpath_here',
...     # Add other necessary XPaths
... }
>>> success = fill_user_form(driver, username, display_name, password, email, group,
↪ quota, manager, NEW_USER_BUTTONS_XPATH)
>>> print(success)
True
```

`utils.user_management.`**`generate_confirm_delete_user_button_xpath`**`()`

Generates the XPath for the confirmation button to delete a user. Assumes the confirmation button has a class that can be uniquely identified and does not depend on the username.

Returns:
: str: The generated XPath for the confirmation button.

`utils.user_management.`**`generate_delete_user_button_xpath`**`()`

Generates the XPath for the user's "Delete user" button based on the given username.

Args:
: username (str): The username to generate the XPath for.

Returns:
: str: The generated XPath for the "Delete user" button.

`utils.user_management.`**`generate_email`**`(`*username*`)`

Generate an email address from a username.

`utils.user_management.`**`generate_name`**`()`

Generate a random name-like string.

`utils.user_management.`**`generate_password`**`()`

Generate a secure password.

`utils.user_management.`**`generate_user`**`(`*group*, *quota*, *manager*`)`

Generates a new user with specified credentials.

Parameters:
: group (str): The group to which the user belongs. quota (str): The data quota assigned to the user. manager (str): The manager of the user.

Returns:
tuple: A tuple containing the generated user's username, display name, password, email, group, quota, and manager.

`utils.user_management.`**`generate_user_actions_button_xpath`**`(`*username*`)`

Generates the XPath for the user's actions button based on the given username.

Args:
username (str): The username to generate the XPath for.

Returns:
str: The generated XPath for the user's actions button.

`utils.user_management.`**`generate_user_xpath`**`(`*username*`)`

Generates the XPath for the user's main div based on the given username.

Args:
username (str): The username to generate the XPath for.

Returns:
str: The generated XPath.

`utils.user_management.`**`modify_quota`**`(`*driver*, *username*, *quota*, *MODIFY_USER_BUTTON_XPATH*, *SAVE_MODIFICATION_BUTTON_XPATH*`)`

Modify the quota for a user.

Parameters

- **driver** (*webdriver object*) – A webdriver object for interacting with the browser.
- **username** (*str*) – The username of the user whose quota is to be modified.
- **quota** (*str*) – The new quota value.

`utils.user_management.`**`navigate_to_users_page`**`(`*driver*, *OPEN_SETTINGS_MENU_XPATH*, *CLOSED_SETTINGS_MENU_XPATH*, *USERS_BUTTON_XPATH*`)`

Navigates to the users page.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. USERS_BUTTON_XPATH (str): The XPath of the "Users" button.

Returns:
None

`utils.user_management.`**`open_new_user_form`**`(`*driver*, *NEW_USER_BUTTON*`)`

Clicks on the "New user" button.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. NEW_USER_BUTTON (str): The ID of the "New user" button.

Returns:
None

`utils.user_management.`**`perform_delete_user`**`(`*driver*, *username*`)`

Deletes a user from the user list.

`utils.user_management.`**`refresh_page`**`(`*driver*`)`

Refreshes the page.

Args:
driver (WebDriver): The WebDriver instance used for browser automation.

Returns:
None

`utils.user_management.`**`test_quota`**`(`*driver*, *username*, *password*`)`

test_quota _summary_

_extended_summary_

Parameters

- **`driver`** (*_type_*) – _description_
- **`username`** (*_type_*) – _description_
- **`password`** (*_type_*) – _description_
- **`quota`** (*_type_*) – _description_

`utils.user_management.`**`wait_then_click`**`(`*driver*, *element*`)`

Waits for the element to be clickable then clicks it.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. element (WebElement): The Selenium WebElement to click.

Returns:
None

`utils.user_management.`**`wait_then_send_keys`**`(`*driver*, *element*, *keys*`)`

Waits for the element to be interactable then sends keys to it.

Args:
driver (WebDriver): The WebDriver instance used for browser automation. element (WebElement): The Selenium WebElement to send keys to. keys (str): The keys to send to the element.

Returns:
None

1.5.6 utils.webdriver_setup module

`utils.webdriver_setup.`**`driver`**`()`

Creates and returns a new instance of the Chrome WebDriver.

Returns:
WebDriver: The Chrome WebDriver instance.

1.5.7 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX