Assignment Part 2 Report

Group 11

We, **Mitkumar Deepakbhai Patel, Dhruv Chotalia** (mention your names), declare that the attached Assignment is our own work in accordance with the Seneca Academic Policy. We have not copied any part of this Assignment, manually or electronically, from any other source, including websites, unless specified as references. We have not distributed our work to other students.

	Name	Task
1	**Mitkumar Deepakbhai Patel**	**Utils tests, Authentication tests, file operation test, Reflection,Report**
2	**Dhruv Chotalia**	**File Integrity test, sharing file test, file operation test, Reflection, Report**

Test Summary:

```
============================= test session starts ==============================
platform linux -- Python 3.12.1, pytest-8.3.4, pluggy-1.5.0
django: version: 5.1.4, settings: secure_file_sharing.settings (from ini)
rootdir: /workspaces/assignng-group11/assignpart2/secure_file_sharing
configfile: pytest.ini
plugins: anyio-4.6.0, cov-6.0.0, django-4.9.0
collected 16 items

tests/test_authentication.py .....                                       [ 31%]
tests/test_file_integrity.py ..                                          [ 43%]
tests/test_file_operations.py ....                                       [ 68%]
tests/test_file_sharing.py ...                                           [ 87%]
tests/test_utils.py ..                                                   [100%]

---------- coverage: platform linux, python 3.12.1-final-0 -----------
Name                                                Stmts   Miss  Cover
-----------------------------------------------------------------------
__init__.py                                             0      0   100%
admin.py                                                4      0   100%
apps.py                                                 4      0   100%
forms.py                                               26      5    81%
migrations/0001_initial.py                              8      0   100%
migrations/0002_alter_fileshare_unique_together.py      4      0   100%
migrations/__init__.py                                  0      0   100%
models.py                                              19      2    89%
utils.py                                               59      7    88%
views.py                                              129     22    83%
-----------------------------------------------------------------------
TOTAL                                                 253     36    86%


============================== 16 passed in 14.34s ==============================
```

Coverage Report:

```
---------- coverage: platform linux, python 3.12.1-final-0 -----------
Name                                                Stmts   Miss  Cover
-----------------------------------------------------------------------
__init__.py                                             0      0   100%
admin.py                                                4      0   100%
apps.py                                                 4      0   100%
forms.py                                               26      5    81%
migrations/0001_initial.py                              8      0   100%
migrations/0002_alter_fileshare_unique_together.py      4      0   100%
migrations/__init__.py                                  0      0   100%
models.py                                              19      2    89%
utils.py                                               59      7    88%
views.py                                              129     22    83%
-----------------------------------------------------------------------
TOTAL                                                 253     36    86%
```

The following sections provide a detailed breakdown of the test cases implemented to validate the functionality of the file-sharing feature. The summarized test execution and coverage reports presented above highlight the overall testing performance and code coverage achieved. Below, each test case is described in detail, outlining the objectives, expected outcomes, test scenario steps, and results. This structured approach ensures a comprehensive understanding of the testing process and its alignment with the application requirements.

1. test_authentication.py

1.1 Test Successful User Registration: This test ensures that a user can register successfully using valid credentials and that the new user is created in the database.

Expected Outcome:

- The user registration request is successful, returning a status code of 302 (redirect).
- A new user with the specified username is created and exists in the database.

Test Scenario Steps:

1. Submit a registration request with valid credentials (username, email, and matching passwords).
2. Verify the response status code is 302, indicating a redirect (likely to the login page).
3. Confirm that the user is successfully created in the database.

1.2 Test Invalid Registration: This test verifies that user registration fails when the passwords do not match.

Expected Outcome:

- The registration request fails, returning a status code of 200 (rendering the same page with an error).
- No new user is created in the database.

Test Scenario Steps:

1. Submit a registration request with mismatched passwords.
2. Verify the response status code is 200.
3. Confirm that the user is not created in the database.

1.3 Test Login Authentication: This test ensures that a user can log in successfully using valid credentials.

Expected Outcome:

- The login request is successful, returning a status code of 302 (redirect).

Test Scenario Steps:

1. Create a test user in the database.
2. Submit a login request with valid credentials.

3. Verify the response status code is 302, indicating successful login and redirection.

3.4 Test Invalid Login: This test ensures that a user cannot log in with invalid credentials.

Expected Outcome:

- The login request fails, returning a status code of 200 (indicating login page with errors).

Test Scenario Steps:

1. Submit a login request with incorrect credentials.
2. Verify the response status code is 200.

3.5 Test Logout Functionality: This test verifies that the logout functionality works as expected and the user's session is cleared.

Expected Outcome:

- The logout request is successful, returning a status code of 302 (redirect).
- The session key _auth_user_id is removed, confirming the user is logged out.

Test Scenario Steps:

1. Create and log in a test user.
2. Submit a logout request.
3. Verify the response status code is 302, indicating successful logout and redirection.
4. Confirm that the user's session does not contain the _auth_user_id key.

2. test_utils.py

2.1 Test Key Generation: This test validates the functionality of the generate_key method to ensure a cryptographic key is generated correctly and saved to the environment file.

Expected Outcome:

- The key is generated successfully.
- The key is of type bytes.
- The key is saved in the .env file under the appropriate environment variable name.

Test Scenario Steps:

1. Call the generate_key method from the FileEncryptor class.
2. Verify that the returned key is valid and of the expected data type.
3. Confirm that the key is saved to the .env file.

2.2 Test File Encryption and Decryption Cycle: This test checks the encryption and decryption functionality for a file, ensuring that file integrity is maintained across both processes.

Expected Outcome:

- Encryption alters the file content.
- The hash of the original file is correctly preserved.
- Decryption restores the file to its original state, and the hash matches the original file.

Test Scenario Steps:

1. Read the content of a sample file and compute its hash.
2. Encrypt the file and verify the content changes.
3. Decrypt the file and compare its content and hash with the original.

3. test_file_integrity.py

3.1 Test File Hash Generation During Upload: This test validates the process of hash generation during file upload. It ensures that the hash generated for the uploaded file is accurate and correctly stored in the database.

Expected Outcome:

- A valid SHA-256 hash is generated for the uploaded file.
- The generated hash matches the expected hash derived from the file's content.
- The hash length is exactly 64 characters (for SHA-256).

Test Scenario Steps:

1. Create and log in a test user.
2. Prepare a sample file with known content and calculate its expected SHA-256 hash.
3. Mock the hash generation process during file upload.
4. Upload the file and verify that the hash stored in the database matches the expected hash.

5. Confirm that the hash length is 64 characters.

3.2 Test File Integrity During Download: This test ensures that the integrity of a file is maintained during download. It verifies that the downloaded file's content matches its original content and hash.

Expected Outcome:

- The file can be downloaded successfully.
- The downloaded file's content matches the original file's content.
- The hash of the downloaded file matches the original hash stored in the database.

Test Scenario Steps:

1. Create and log in a test user.
2. Prepare a sample file with known content and calculate its expected SHA-256 hash.
3. Mock the encryption and decryption processes for the file.
4. Simulate the file upload and download process.
5. Verify that:
    - The server returns a 200 status code for the download.
    - The file's content matches the original content.
    - The downloaded file's hash matches the expected hash.
6. Clean up any temporary files created during testing.

4. test_file_operations.py

4.1 Test File Upload: This test ensures that authenticated users can upload files successfully. The uploaded file is encrypted, and its hash is stored in the database.

Expected Outcome:

- The file upload request is successful, returning a status code of 302 (redirect).
- The file is saved to the database with the correct user association.
- The file hash is correctly generated and stored.

Test Scenario Steps:

1. Create and log in a test user.
2. Prepare a test file for upload.

3. Mock the encryption process to simulate hash generation.
4. Submit the file upload request.
5. Verify the file is saved in the database and the hash matches the mocked value.

4.2 Test Unsupported File Upload: This test verifies that unsupported file types cannot be uploaded.

Expected Outcome:

- The file upload request for unsupported types fails, returning a status code of 302 (redirect).
- No file is created in the database.

Test Scenario Steps:

1. Create and log in a test user.
2. Prepare an unsupported file type (e.g., .exe).
3. Attempt to upload the unsupported file.
4. Verify the file upload is rejected, and the file does not exist in the database.

4.3 Test File Download: This test ensures that a file owner can download their file successfully, with decryption applied during the download process.

Expected Outcome:

- The file download request is successful, returning a status code of 200.
- The downloaded file content matches the decrypted content.

Test Scenario Steps:

1. Create and log in a test user.
2. Create a test file and save it in the database.
3. Mock the decryption process to simulate the file content after decryption.
4. Submit the file download request.
5. Verify the response status code and downloaded file content.

4.4 Test Delete File: This test ensures that a file owner can delete their uploaded file successfully.

Expected Outcome:

- The file deletion request is successful, returning a status code of 302 (redirect).
- The file is removed from the database.

Test Scenario Steps:

1. Create and log in a test user.
2. Create a test file and save it in the database.
3. Submit the file deletion request.
4. Verify the file no longer exists in the database.
5.

5. test_file_sharing.py

5.1 Test File Share

Description:
This test ensures that authenticated users can share files with other registered users.

Expected Outcome:

- File sharing request is successful, returning a status code of 302 (redirect).
- A new record is created in the FileShare model linking the file to the recipient.

Test Scenario Steps:

1. Create two test users: file owner and recipient.
2. Log in as the file owner.
3. Create a test file owned by the file owner.
4. Share the file with the recipient via a POST request.
5. Verify the sharing record exists in the FileShare model.

5.2 Test Duplicate File Share

Description:
This test ensures that sharing the same file multiple times with the same recipient does not create duplicate entries in the FileShare model.

Expected Outcome:

- The first share request is successful, returning a status code of 302 (redirect).
- The second attempt returns a status code of 200 (indicating the request is blocked or handled gracefully).
- Only one record exists in the FileShare model.

Test Scenario Steps:

1. Create two test users: file owner and recipient.

2. Log in as the file owner.
3. Create a test file owned by the file owner.
4. Share the file with the recipient via a POST request.
5. Attempt to share the file again with the same recipient.
6. Verify only one sharing record exists in the FileShare model.

5.3 Test Unauthorized File Access

Description:
This test ensures that users who are not explicitly granted access to a file cannot download it.

Expected Outcome:

- Unauthorized users attempting to download a shared file are denied access, with a status code of 302 (redirect to an error or warning page).

Test Scenario Steps:

1. Create three test users: file owner, authorized user, and unauthorized user.
2. Log in as the file owner.
3. Create a test file owned by the file owner.
4. Share the file with the authorized user.
5. Log in as the unauthorized user and attempt to download the file.
6. Verify the unauthorized user is denied access.