File Upload

Developer: Allison Austin

Submitted March 7, 2023

Reviewer: Joshua Quibin

Reviewed March 7, 2023

Criteria

Major Positives & Negatives

Positives

- Sequence diagrams include both frontend and backend validation for file type/file size and file ownership, along with S3 Bucket validation.
- Sequence diagrams include detailed logging, message prompts, queries, method signatures, axios calls, return values, HTTP response codes, and S3 Bucket requests.
- Sequence diagrams follow consistent formatting with cascading request/response structure.
- Design follows MVC architecture, utilizing a view (FileView), model (FormFile), and controller (FileController).

Negatives

- The design documents fail to address all requirements outlined in the BRD. In the BRD, the functional requirements specify that users can download files from group storage to their local device. The design documents fail to outline the file download functionality entirely.
- My research indicates that uploading an object with a pre-existing key name to an S3 bucket will simply overwrite the previously uploaded file if versioning is disabled. (Uploading Objects - Amazon Simple Storage Service, 2023) / (*Enabling Versioning on Buckets - Amazon Simple Storage Service*, 2023) With no validation for uploading files with the same naming conventions, we are potentially at risk for data loss with overwritten files.

- There are no implementation details outlined surrounding the expedited process of permanently deleting a file in the case that a user does not want their files to exist within our system or the option to restore their file after initial deletion.

Unmet Requirements

- *Files can be downloaded from storage to a user's device. (githelsui/WeCasa: @ Team HAGS JP.)*
    - The diagrams fail to outline the file download functionality. Hence, this requirement is not met by the design.

Design Recommendations

- Downloading a file from storage to a user's device is a functionality that I believe elicits its own separate design documentation, including sequence diagrams outlining both failure/success scenarios and accompanying wireframes that document the process' flow. This functionality would require minimal validation, aside from Group Access validation, given that this functionality isn't permission exclusive to only the file uploader as outlined in the BRD.
    - Positives: Given that you have pre-existing documentation for viewing all group files, you can use this as a baseline as these processes essentially go hand-in-hand, saving you time with flushing out the design documentation for file download.
    - Negatives: N/A
- In the case of having the user upload an object with the same name, my design recommendation is to have versioning enabled for the S3 Bucket which enables users to upload duplicate files with similar naming conventions as pre-existing files, and cuts down on the need for name validation.
    - Positives: Easiest option to implement, and this eliminates any worry surrounding data loss issues with file overwrites.
    - Negatives: The opportunity for duplicate files to exist within group storage could prove to be confusing for the end user from an organizational standpoint. This can be remedied by including name validation and simply keeping versioning enabled as a failsafe for data loss.

- Alternatively, if we decide to keep versioning disabled for the S3 bucket, file name validation should be a mandatory inclusion for file uploads and we can moderate the option for users to overwrite files that belong to them by prompting them with a confirmation message.
    - Positives: Cutting out the option for duplicate files seems to be the more organized approach, minimizing confusion amongst end users.
    - Negatives: Harder to implement, as this would require additional validation/testing, and we also risk data loss with overwritten files. This can be remedied if we implement some form of version control where overwritten files can be restored similarly to deleted files.
- Despite not being outlined in the BRD, I believe we should provide the user with the option to permanently delete their files as opposed to only giving them the option to recover a deleted file, in the event that they want to expedite the process of removing a file from our system.
    - Positives: Adherence to the California Consumer Privacy Act's regulations surrounding the *Right to Delete. (California Consumer Privacy Act (CCPA), 2018)*
    - Negatives: Requires updating the design documentation along with additional testing/validations.

Test Recommendations

Integration Tests

- File Upload/Delete: To test uploading/deleting a file, I would have a pre-existing file that passes validation (is less than or equal to 10 MB file size and of a valid file type), and upload and delete the file to and from the system and make sure that the file gets updated and/or deleted for every user who shares that group space.
    - Arrange the test file to update and delete. Make sure there are at least two users in the same group who will have access to the file.
    - After uploading the file, call GetGroupFiles on the group_id of the two users and check that the File objects are the same.
        - Design a separate test that checks whether the file upload exceeds the group's overall storage capacity (15 GB)
    - Delete the file and call GetGroupFiles on the group_id of the two users.
    - Check that changes are reflected for the deleted file for both users.

- Recover File: To test recovering a file, I would manually create a deleted file within a valid time frame less than 24 hours.
    - Arrange the test file to be recovered. Make sure there are at least two users in the same group who have access to the file and ensure that the test user is the owner of the file and call RecoverFile().
    - After recovering the file, call GetGroupFiles on the group_id of the two users and check that the recovered File object appears.
- Permanently Delete File: To test for old/deleted files being properly removed from the database, you can manually create files and run the scheduled events.
    - Arrange a day-old file to be deleted.
    - Run the scheduled job manually by using the command line.
    - Check the system and test that the files do not get returned from the GetGroupFiles() return object.

Unit Tests

- Input Validation: To test for input validation, I would attempt to upload files to the system that do not meet the following requirements:
    - >= 10 MB File Size
    - Photo File Type: PNG, JPEG, and GIF
    - Document File Type: PDF, DOC, DOCX, HTML, TXT
- Permission Validation: To test for permission validation, I would attempt to delete a file as a user who does not have ownership permissions

References

California Consumer Privacy Act (CCPA). (2018, October 15). State of California - Department of Justice - Office of the Attorney General. https://oag.ca.gov/privacy/ccpa#:~:text=Right%20to%20delete%3A%20You%20can,required%20to%20keep%20the%20information).

Enabling versioning on buckets - Amazon Simple Storage Service. (2023). Amazon.com. https://docs.aws.amazon.com/AmazonS3/latest/userguide/manage-versioning-examples.html#enable-versioning

githelsui/WeCasa: @ Team HAGS JP. (2023, Feb 17) *GitHub* https://github.com/githelsui/WeCasa/blob/main/docs/BRD%20v2.0.pdf

Uploading objects - Amazon Simple Storage Service. (2023). Amazon.com. https://docs.aws.amazon.com/AmazonS3/latest/userguide/upload-objects.html