

# Final problem statements for easy copy-paste

## SDE1

Design and implement a Simple document Service where users can create documents and read the same.

A document has a name and associated string content. Document=> <name{string}, content{string}>

1. All documents are private when created.
2. Owners of documents can {grant} {read OR edit} access to other users
3. Only the owner can delete a document
4. Username will be just a string. Every action like create/read/edit/delete must be made on behalf of a user

## SDE2

Design and implement a Simple document Service where users can create documents and read the same.

A document has a name and associated string content <name{string}, content{string}>

1. All documents are private when created.
2. Owners of documents can grant access to other users
3. Grants can be made at global level as well. For example, if access is granted globally, then every user should have access to that document.
4. Username will be just a string. Every action like create/read/edit must be made on behalf of a user
5. Have different tiers. Hot tier reads should be served from memory. Cold tier reads should be served from the disk. Owner can specify which tier

## SDE3

Design and implement a Simple document Service where users can create documents and read the same.

A document has a name and associated string content <name{string}, content{string}>

1. All documents are private when created.
  2. Owners of documents can grant access to other users
  3. Grants can be made at global level as well. For example, if access is granted globally, then every user should have access to that document.
  4. Username will be just a string. Every action like create/read/edit must be made on behalf of a user
  5. Have different tiers. Hot tier reads should be served from memory. Cold tier read should be served from the disk. service should be able to bring the globally frequently accessed documents into the hot tier.
  6. **[Discussion only, time permitting]** -> user must be able retrieve any of the last N versions of a document
-

## Notes:


1. As its interview, create/read/update can be in memory and don't need to read/write to disk[**for sde1 only**].
2. Everything can be in the same process, no need for separate client/server etc. No need to use frameworks.
3. Separation of concerns/DRY: do they keep authorization checks separate from app logic? Is there code duplication?
4. Error handling should also be present, in case of delete operations an Invalid Operation/Permissions denied error has to be thrown. Similarly for read/edit operations on a document without permission
5. Another aspect to consider is concurrent access to documents. Does the candidate identify this aspect? Should reading be allowed if another update is in progress? No need to implement, but plus points if they can think of it.
6. What if 2 users upload documents with the same name? Does the candidate identify/handle this? How to handle when such documents are granted by 1 user to another user.
7. It is OK if the solution is not algorithmically optimal.[eg: frequently accessed documents]
8. It is OK to google for file open/close/read/write if the candidate is not familiar with file handling functions in their language

## Suggestions from walkthrough session

1. Can keep a list of users in the system at the start. Check for error handling when document is made public, but user does not belong in this user list
2. Instead of actually writing to disk, can have 2 implementations, both in memory. First simulates a fast memory path[hot tier]. Second simulates slow disk path[cold tier]

## Expectations

Taken from [evaluation matrix](#)

Expectations	SDE1	SDE2	SDE3
Requirements + Low Level Design			
Asks Relevant Questions		✓	✓
Narrowed down scope		✓	✓
Identified Multiple solutions and reasoned one solution		✓	✓
Identified edge cases			✓
Flow walk through			✓
Coding			
Optimum solution	✓	✓	✓   
Readable Code	✓	✓	✓
Modular Code		✓	✓
Handles edge cases		✓	✓
Extensible Code			✓
Quality			
Showcases happy path scenario	✓	✓	✓
Showcases few edge case scenarios		✓	✓
Identify and fix issues in quick time			✓
Extensibility			
Inject new features with minimal code changes			✓
Ease of identifying and fixing issues after new features		✓	✓

Sde1

1. Functionally works/correct.

2. Plus points if able to implement 1 “small” extension with minimal changes[Eg: change in-memory implementation to disk based implementation without changing all the core functions]
3. Interviewer can try to limit scope wherever possible

## Sde2

1. Sde1 + asks clarifications +HotTier/Cold Tier
2. Reusable+DRY wherever possible.
  - a. For eg: code to check “read access” vs “write access” will be similar. Do they repeat themselves or abstract into a common function
  - b. Plus points if they can separate the logic related to access checks, storage into different functions/classes.

## Sde3

1. SDE2 + Versioning support -> deep-dive on approaches and how it can be added in current implementation. Implementing this is not expected. but a discussion can be done.

## Sample solution

[https://github.com/razorpay/armory/tree/interview\\_document\\_service/interview\\_document\\_service](https://github.com/razorpay/armory/tree/interview_document_service/interview_document_service)

This is an example only. Consists of an example skeleton + implementation for some functions.  
This is not the only approach and is for reference only.

## Not recommended approaches

1. Every aspect(access check, storage) in 1 function

```

@Override
public Boolean updateDocument(Owner owner, String documentName, String documentContent) {

    if(owner== null || documentName == null)
    {
        return false;
    }else {

        if (Data.accessType.get(documentName).equalsIgnoreCase("public")) {
            Data.data.put(documentName, documentContent);
            return true;

        } else {

            if (owner.writeList.contains(documentName)) {
                Data.data.put(documentName, documentContent);
                return true;

            } else {

                return false;

            }

        }

    }

}
}

```

a.

## Extensions

PS: not all extensions are expected to be implemented

## Large

1. Have different **tiers**. Hot tier should be served from memory. Cold tier should be served from the disk. Edge cases can be around how are updates handled
  - a. SDE2 -> Owner can specify which tier
  - b. SDE3 -> service should be able to bring the globally frequently accessed documents into the hot tier.
2. ~~Instead of just operating at the level of "document", can add the concept of "directory". "Directory" contains "documents" and possibly other subdirectories.~~
  - a. ~~Operations on directories can include create/delete/list.~~
  - b. ~~Can steer discussion around how grants/block can be made on directories, whether those are carried to the documents within the directory etc.~~
  - c. ~~There is no right or wrong approach, but just that it's good if the candidate can think of them before implementing.~~
3. **Versioning support** - user must be able retrieve last N versions of a document

- a. How will the candidate modify solution to support versioning(retrieval of previous version of same document)
- b. Can the candidate come up with different approaches?(eg: maintaining the entire document vs storing the diffs only. Does the candidate evaluate the tradeoffs involved).
  - i. If given an interface to generate diffs, merge diffs, can the candidate implement versioning?

## Small

4. **[SDE1 Extension]** Instead of in-memory, we needed durable storage, what is the scope of change needed? Is the change localized to a specific class/module or will it require a refactor of the entire codebase?
5. ~~Time can be added as a dimension. For eg: grants are valid for "x" duration only and after that should be ignored~~
  - a. ~~Plus points if they test this by injecting time instead of adding sleep() in the code.~~