Design and Development of a Cloud Storage Application for Advanced Programming Subject

Juan Nicolas Diaz Salamanca
20232020059
Faculty of Engineer
Computer Science
District Jose Francisco de Caldas University Bogota DC
Email: jndiazs@udistrital.edu.co

Mathew Zahav Rodriguez Clavijo
2023202050
Faculty of Engineer
Computer Science
District Jose Francisco de Caldas University Bogota DC
Email: mzrodriguezc@udistrital.edu.co

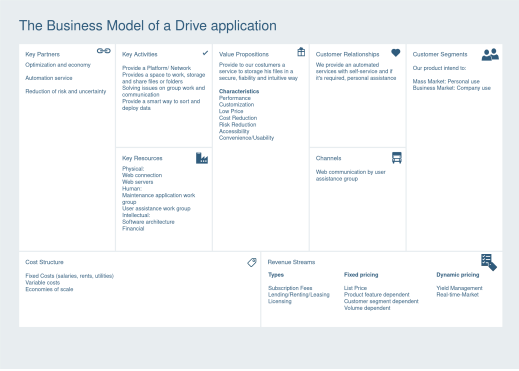
***Abstract*—This document describes the process of developing a cloud storage web application, from identifying user needs to deployment. It focuses on creating quality software, following design principles such as Single Responsibility, Open/Closed, Liskov Substitution, among others, to ensure a robust and flexible system. The application architecture seeks to be scalable, maintainable, and reliable. The development is based on user surveys and the definition of user stories, which guided the creation of the requirements and the design of the interface, resulting in a solution adapted to the user's needs.**

1. Introduction

The following document will describe the process of developing a cloud storage web application, starting from the user's needs to the implementation in code and execution; in order to develop quality software.

1.1. What is a cloud storage application?

Cloud storage applications are applications in charge of providing cloud storage services, that is, providing the user with the ability to store files, organize them in folders and access them from any computer with an internet connection. The following diagram is the business model of a cloud service company:



1.2. What is quality software?

Given that these are the functionalities of cloud storage software, then it is relevant to consider how we can generate quality software with these functionalities; starting from the author Bertrand Meyer, it is software that meets the following characteristics:

- Correctnes: "A software must execute its tasks as stated in its specifications"[1]
- Robustness: "The ability to operate in as many conditions as possible" [1]
- Extensibility: "The ease of adapting to changes in your specifications" [1]
- Reusability: "The ability to be used in a wide variety of contexts" [1]
- Compatibility: "The ability of the software to be used by other software" [1]
- Portability: "The ability of software to work in various hardware and environments" [1]
- Ease of use: "The ability of any user, regardless of their knowledge, to use the software to its full extent" [1]
- Timeliness: "The ability of the software to launch itself at the moment the user needs it" [1]

1.3. Design Principles

In order to comply with these characteristics, the following principles have been used to take into account when planning, designing and coding a software

- Single responsibility principle (SRP): "A computer system varies only to satisfy one type of customer and collaborators, these is its reason for change"[2] The above definition is more appropriate since SRP is often misinterpreted to assume that each object within a system should only perform one task or functionality. In contrast, SRP states that an object can have as many functionalities, but they must all be directed and consumed by the same type of client.

- Open and Closed Principle (OCP): ”The behavior of a software artifact must be understandable without needing to modify it”[2] This principle is focused on guaranteeing the continuity of the software, that is, that the software with respect to time is in a constant increase of its functionalities. The greater the modifications within the code, the less extension of the original code can be made and the less continuous the software will be. This problem is attacked by the OCP principle, always proposing to avoid modification and replacing it with extension.
- Liskov Substitution Principle (LSP): ”If for every object s1 of type S, there is an object s2 of type T and p is defined in terms of T; then the behavior of p is the same for S1 or S2 because it is a subtype of T.”[2] This principle provides a rule to guarantee the extension of the software sought by OCP, since, to extend the functionalities of an object, instead of changing it, we can create a subtype of that object by means of inheritance. In this way we guarantee OCP but LSP tells us that this action must always take into account that the subtype must have all the attributes and functionalities of the parent class so that it can be used in contexts in which the parent class is required.
- Interface Segregation Principle: ”No software component should be forced to implement and rely on modules it does not need”[2] Following the goal of a single client type as the change reason for a class, ISP proposes a way to implement this solution in code, separating the object interfaces from different change reasons
- Dependency Inversion Principle: ”The most flexible systems are those where module dependencies are aimed only at abstractions”[2] ISP proposes to separate interfaces from objects; Interfaces are classes that lack functionalities but have a series of them declared, so every time a class wants to inherit from them, it must at least implement all the functionalities that the interface dictates and for a class that wants to consume its functionalities, it cannot consume any of the interface because they are empty but for this reason, The class it consumes has the guarantee that every class it inherits from the interface will give logic to that declared functionality and consequently will be able to treat all objects in that interface in the same way even if they have opposite logics. This flexibility is only achieved when all objects depend on abstractions.

1.4. Architecture

Following the suggestion of Robert C Martin, ”uncle bob”, once the characteristics that each class meets and related following the design principles have been raised, it is possible at that moment to visualize the composition of everything as a system, this structure is known as architecture defined as ”The architecture of a software system is the shape given to that system by those who build it. (...) The purpose of that shape is to facilitate the development, deployment, operation and maintance of the software system”[2]. In this way, an architecture must be:

- Scalable: The ability to always respond as efficiently as possible to requests from any number of clients
- Maintainable: The ease of extending and adapting the architecture over time to the needs of the users
- Reliable: The ability to perform to your specifications in the widest variety of situations.

2. Methodology: Concept Design

2.1. User stories

For the development of a cloud storage application, a survey of potential customers and the needs they have within this software was carried out. These needs were summarized in the following user stories:

- I as a user I expect an intuitive software to upload and sort files.
- I as a user, I except to keep my files and folders secure of thirds persons.
- I as user, I except that my storage always be enough for my folders and files.
- I as user, I except a graphic and intuitive interface.
- I as user, I expect to open my files with third party applications.
- I as user, I expect to download files from my storage to my personal
- computer.
- I as user, I expect to see the modification date of my files and open it
- without required to download.
- I as user, I expect to sort my files by name and modification date.
- I as user, I expect an easy access and search through my storage.

From these user stories, the following functional and non-functional requirements were defined:

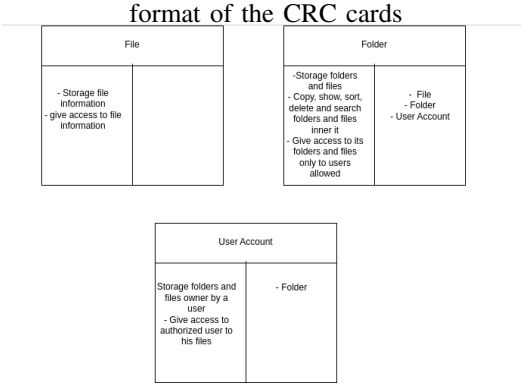
2.2. Requirements

- Browser: The applications must include a search functionality to get files and folders that match with user input.
- Elements visualization: The application must show folders and its files with modification date, name, extension and size.
- Pre-visualization of files: The application must allow to open files without required the download in the user pc.
- Organize elements: The application must allow to sort folders and files by his name and modification date.

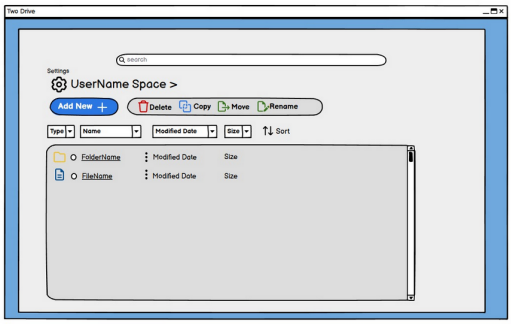
- Friendly GUI: The application must supply and friendly user interface for the user activities.
- Compatibility: The application must work with any application which can open and edit files inner it.
- Security: The application must not give unauthorized access to files and folders from a user
- Storage Consistency: The application must keep the integrity of the user files.

2.3. Abstraction

Bases on requirements, the following abstraction of the entities that meet the given needs was made, using the format of the CRC cards

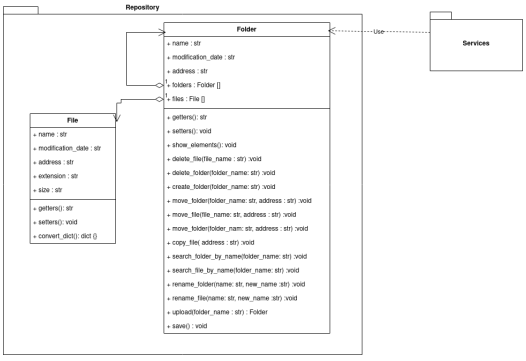


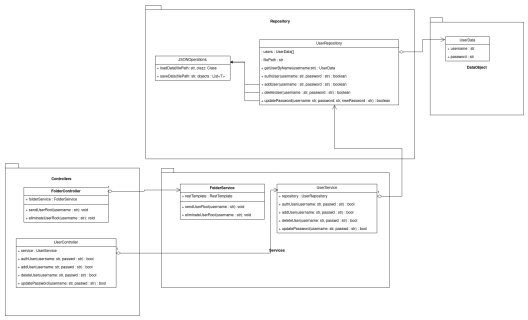
In addition, a first design of the graphic interface was made through a mockup:



3. Technical Design

Based on the conceptual design, the technical design presented below was carried out





These design are based on microservices architecture, separate on:

- Data or Data Objects : Bondaury objects that its only responsability is storage data
- Repository: Objects which manage the Data or Data Objects in order to access to its data, change it or delete it.
- Services: Objects which summary the repository funcionality in order to give more accesability to clients.
- Controller: Objects whichs transforms services into web services.

These two backends works separately without any dependencies, In this way we can secure that if one backend has errors it will not affect the other and as long as the frontend keep connected with the backends at the same time, it will not afected neither.

4. Conclusion

The methodolody proposed has provided a successfully way to understand whats clients needs and wants into a complete operate software architecture; following the steps described at design, we could be a step foward to guarantine a quality software of any type, as a example show it, the cloud storage applications

Acknowledgments

The authors would like to thank to all people which participate answering the survey to understands their needs of a cloud storage application.

References

[1] B. Meyer, *Object-Oriented Software Construction*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 1997.

[2] R. C. Martin,*Clean Architecture: A Craftsman's Guide to Software Structure and Design.*, Boston, MA, USA: Prentice Hall, 2017.