MANİSA CELAL BAYAR UNIVERSITY

CSE 3105/ CSE 3137

OBJECT ORIENTED ANALYSIS AND DESIGN

FALL 2020

COURSE PROJECT: *Media Browser Application*

System Design Document

Group 12

Oğuzhan Yavuz – 160315036

Murat Can Yılmaz – 190315079

Emre Tunç – 160315014

Arda Dumanoğlu – 190315072

17 January 2021

Table of Contents

1 Introduction

While we were developing our project we focused on reliability, flexibility, a bit of performance and specially usability. As we mentioned in RAD, our system is easy to use so it should be able to resolve complexity. We want to integrate people with new trends and technologies. We wanted to design something new but in order to ensure transition we kept the traditional methods.

1.1 Purpose of the System

The main goal is decreasing system requirements and display any media files without an error.

Our main goal in this project is to perform better than its previous variations and provide a better user satisfaction. While providing performance we make sure that our system is secure. Users can store their data in a cloud storage so we must protect their privacy.

1.2 Design goals

1.2.1 Readable

We want to ensure that other developers or users understand our system's operation clearly. Because of unreadable code causes bugs or duplicated codes, we make our system as simple as possible.

1.2.2 Reliability

User's information and access to cloud storage is controlling by a short code(6-digit). Our System intervenes immediately in case of any crash or attack and protects user data.

1.2.3 User Friendliness

The Media Browser App has an easy-to-understand user interface. It is easy to learn so everyone at all ages can use it without instructions.

1.2.4 Robustness

For every user system information and usage reports are backing up every 15 or 30 days (Users can change). Thus, we have full control on our system. We use an encrypted connection to ensure a secure connection during backup.

1.2.5 Flexibility

We are aware that the system has to improve in the future. Thus, we make sure that some subsystems can be converted into desired shape. If any requirement changes in the future, system can adopt easily. Also, the App runs with almost all audio codec and file extensions.

1.2.6 Security

The system securely stores user information and files. Without a permission nobody can access a user's data.

1.2.7 Extensibility

If the system needs to extend, our system design easily provides it. You can always add new functions, attributes, or modify any existing subsystem. These extensions can lead this project into another level.

1.2.8 Open Source

This one is very important to us. Developing an open source program is important in terms of being an example for software developers who are new or who want to develop in this field. Software developers can change or use our program in any way they want. Also, this makes our system more secure. We all make mistakes and if we did any mistakes a good programmer can detect it and we can intervene the mistake.

1.2.9 Ease of Learning

We used non-technical terms in our user interface. Users can easily find what they wanted specifically.

1.3 Trade Off

1.3.1 Functionality vs Usability

In our project, we considered usability is more important than functionality. We aimed a clearer and a simpler experience for users.

1.3.2 Cost vs Robustness

We intend to cost less as much as possible, but system's robustness is also important for us. We don't want errors or bugs in our system to became more secure and impeccable. This comparation is hard to us because robustness seemed like costs too much. However, if you build a less durable system consequences may be more serious. That is why we considered robustness rather than cost. We want our system to have a solid developing process.

1.3.3 Rapid Development vs Reliability

The purpose of the system's being open source was to build it solid. We wanted to build this system as fast as we can like every programmer, but we don't want to put less emphasis on planning. System has to continue to be developed in the future, but we can't rush our development plan. Therefore, system's reliability outweighs its rapid development.

1.3.4 Efficiency vs Compatibility

We all know that a system's efficiency or effectiveness is important. However, we agreed on efficiency is not more important than compatibility. We wanted that our system can work on any operating systems, almost on any device. Our system is not a one-placed system so it should be compatible with many systems.

1.3.5 Readability vs Backward Compatibility

This is as hard as cost vs robustness. Backward compatibility means that system can use the same data and equipment smoothly. This is important because we planned that with regular updates some changes may occur. System must be compatible with its older versions. However, our system is open source and other developers should be able to understand clearly. This is a 50/50 to us because we wanted both to work without difficulty.

2 Current Software Architecture

Most of today's media browsers do not support cloud storage system. A media browser that does not support the cloud system is useless in today's world. Our system will benefit its users with this feature, as well as the other features. Its advanced and simple interface will provide ease of use.

3 Proposed Software Architecture

3.1 Subsystem decomposition

Our system is supported with various subsystems. Subsystems which connected to each other, provide services to each other on the same level of abstraction. We choose client-server architecture because we decided this architecture is the most suitable to our system. We have 2 server subsystems and they are connected. Although this application works on the device, we have a cloud subsystem that makes the difference.

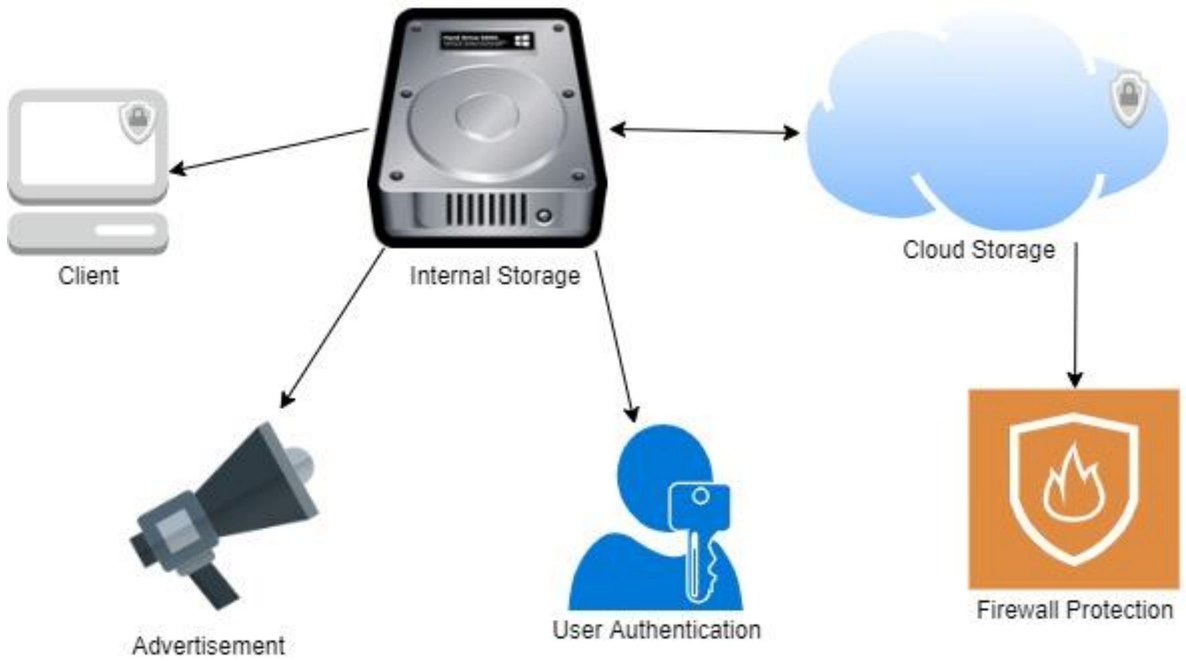

Figure 1: Architecture of the system (Client – Server)

Client-server architecture enables data to be collected in a single center and more secure. The advantages of a client-server architecture are greater security of the system, more control on network traffic passing through the network. The control on network traffic means, being able to see what each user is doing, and we can limit their certain actions on system. This action protects the system against viruses spreading or attacks from other devices. Also, the server can provide much more storage space than most storage devices.

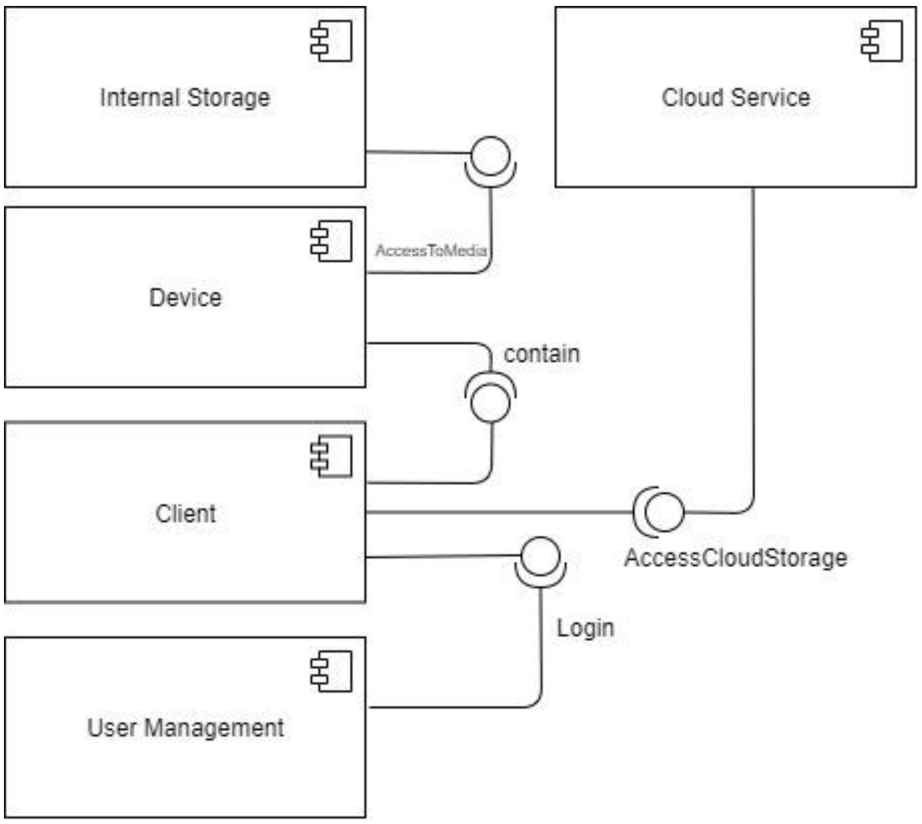

Figure 2: UML Component diagram (SW Architecture)

Client Subsystem: This subsystem is responsible for creating a bridge between the user and the developer.

Storage Subsystem: We have 2 different storage subsystems. One of them is internal storage of the device which allows user to display any media stored in device. The other one is cloud storage system which allow user to display any media that backed up a cloud storage.

Advertisement Subsystem: This subsystem is responsible for the advertisements.

User Authentication Subsystem: This subsystem is responsible for verifying the login information of users which is required for accessing the system

Firewall Protection Subsystem: Is responsible for the protection of the cloud storage subsystem. Tries to prevent any attack or interruption.

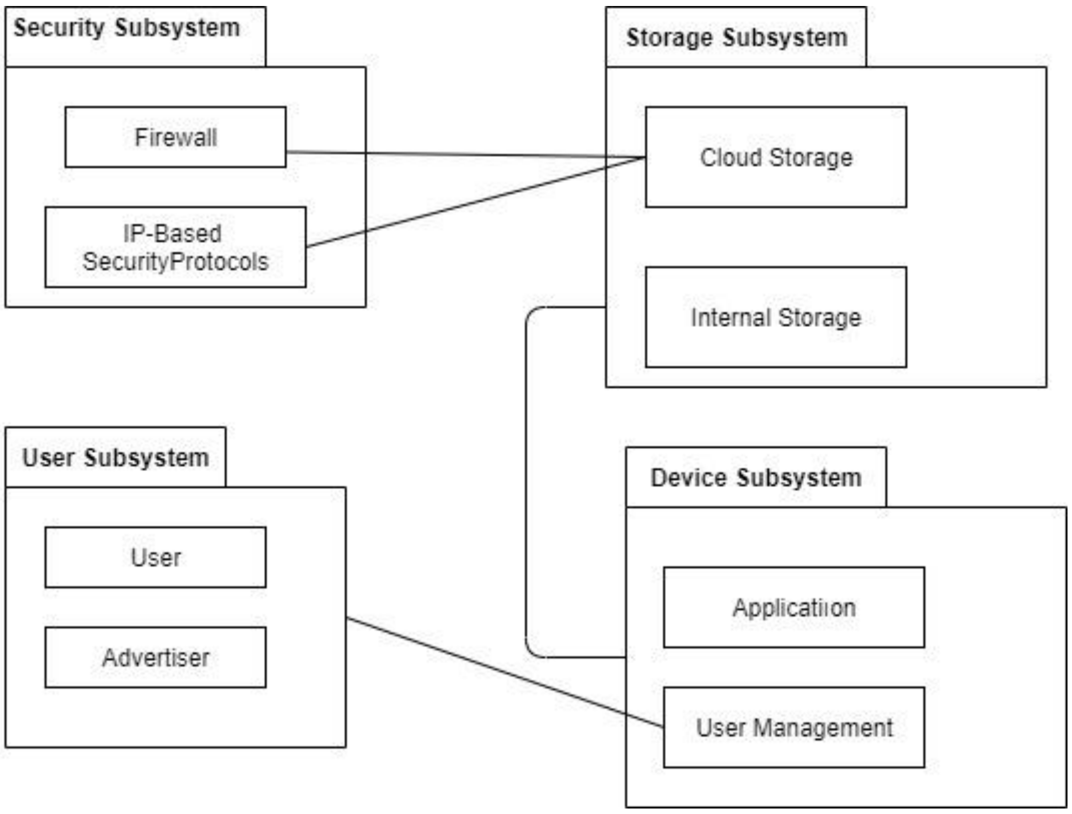

Figure 3: Subsystem Decomposition

3.2 Hardware/software mapping

We choose client-server architecture to assist users, to have access to their remote storage securely. The system is distributed as users accessing their own data or remotely access cloud server storage data from their devices.

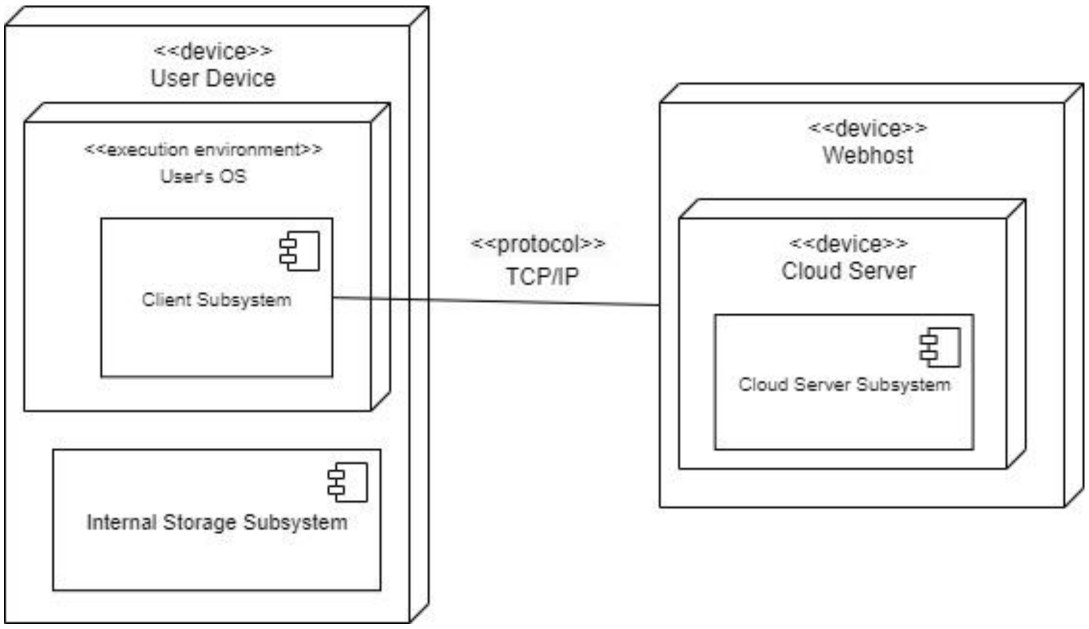

Figure 4: Hardware – Software Mapping

3.3 Persistent data management

For our app we chose SQL database. The SQL language is well known, with many functions and a concise syntax. Having been around for many years, and being widely used in many enterprise applications, there is a wide range of community and enterprise support for SQL databases. There are also many different databases that are of an enterprise standard.

With support for indexes and lots of other optimizations, SQL databases are the fastest for carrying out searches of data over a single table. Performance can drop off when searching across multiple tables, so a good design is important to maintain performance. And in our app, we have only 4 tables so this shouldn't be a problem either.

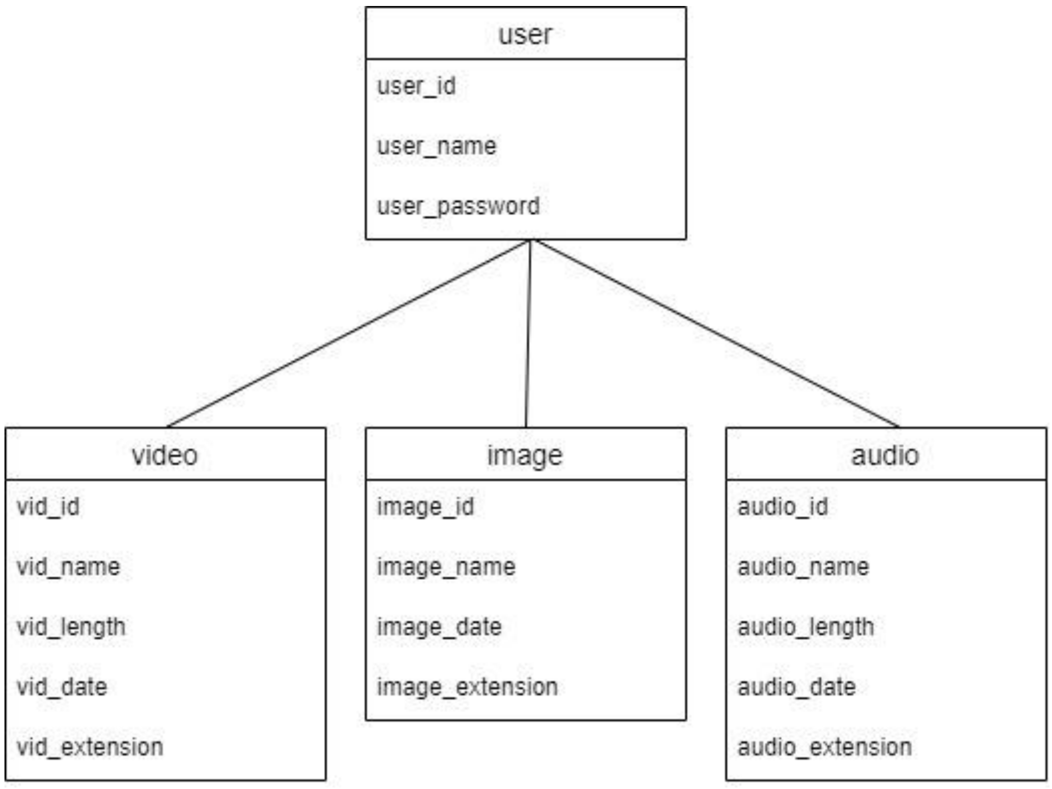

Figure 5: Database Schema

Due to foreign-keys and the maintaining of referential integrity between tables, SQL databases can retrieve data across several tables quickly. The SQL language also has good support for this using JOINS. However as stated above a good design is important to maintain performance, as well as optimized queries.

The worst con of the SQL database is adding, deleting the rows and columns of the tables but in our app, we will never need this.

3.4 Access control and security

3.4.1 Authentication

In this section, access and security matters have been covered through access matrix below. There are two actors in the access matrix, and they have some access rights to relevant classes. Access matrix is very easy to understand. What each actor can do in the relevant classes and which methods can be used are clearly stated. Our system's authentication is first step to pass for using the program. Authentication operation checks the user whether the user exists or not.

3.4.2 Authorization

This is the action phase, in this phase, users are confirmed. Users can play any media by themselves or they can host/join a party. Also, they can access cloud service. Users can only upload and download their own medias.

	Media Browser App	Registration	Party	Cloud Service

User	playMedia()
	editImage()
	sortMedia()
	searchMedia()
	managePlaylist()
		signIn()
		signUp()
			hostParty()
			joinParty()
				upload()
				download()
				backup()

Advertiser	advertise()	signIn()
		signUp()		

Figure 6: Access Matrix

3.5 Boundary conditions

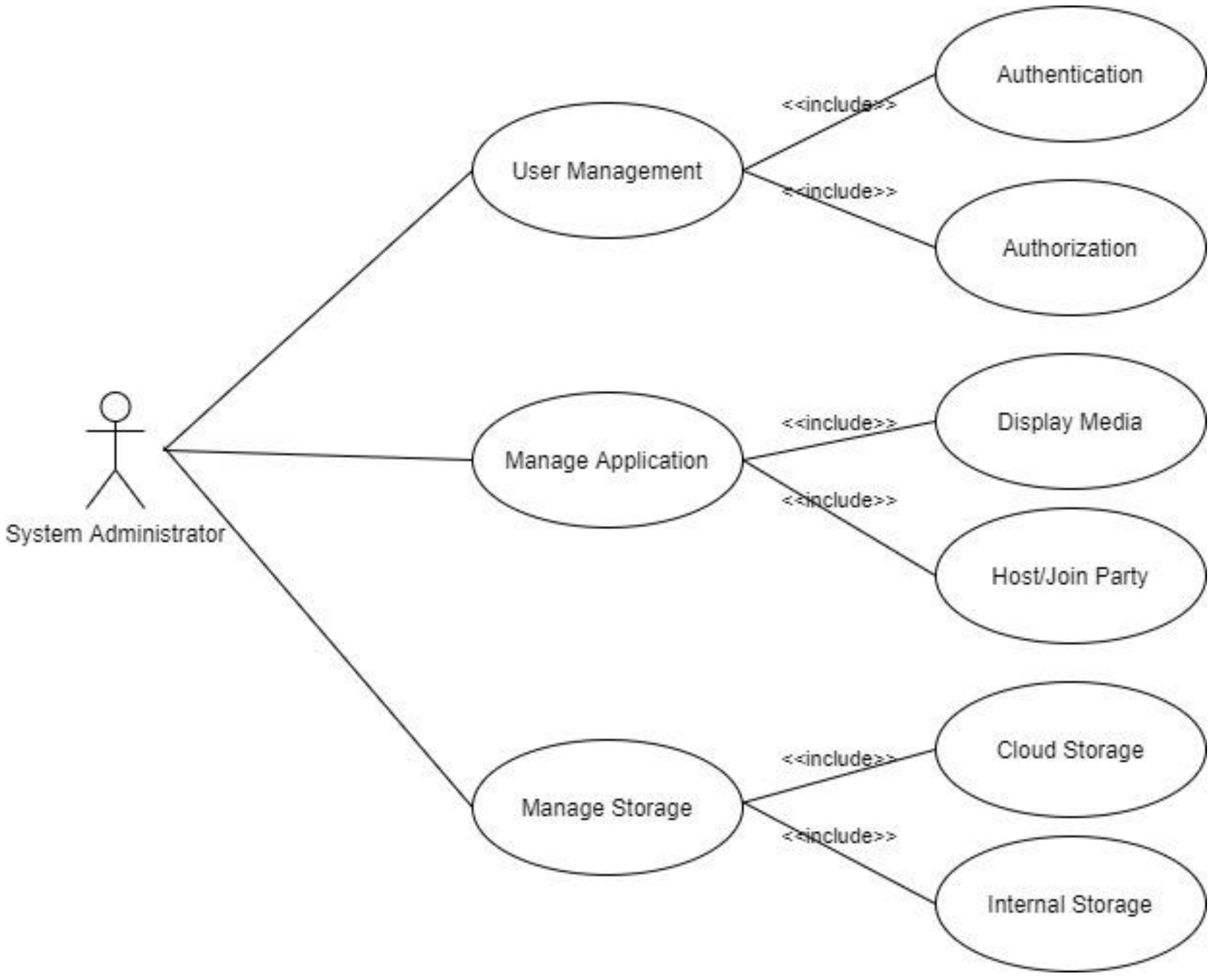

Figure 7: Boundary Conditions

User Management: In order to register to the system, data which user submitted must match with data in the system. Confirmed users can use the system.

Manage Application: This use case consists 2 part of usage. These two usage methods are: users can display their media files on their own or they can host/join a party to surf the media together.

Manage Storage: This use case explains users can access their device's internal storage or a cloud storage that application provides. When the connection is lost between cloud system and the device displaying the media should be shut down.

4 Subsystem Services

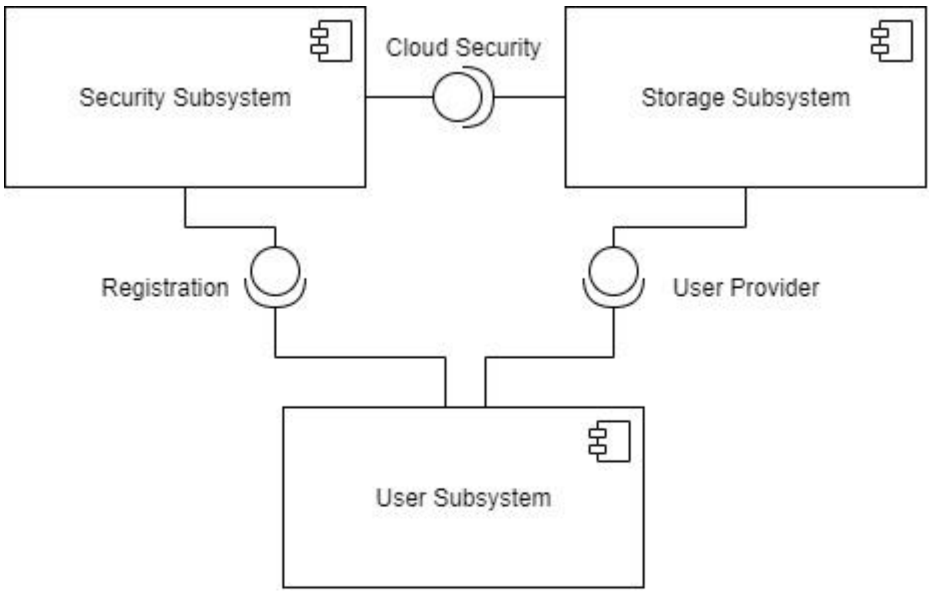

Figure 8: Subsystem Services

Security Subsystem: This subsystem provides security to cloud storage access. Also, it holds the registration data.

Storage Subsystem: It holds both internal storage subsystem and cloud service subsystem. When a user sends data request storage subsystem provides data to the user.

User Subsystem: This subsystem is the subsystem that users interact with. This subsystem allows users to use the application. User Subsystem contains user interfaces, party organizations etc.

5 Glossary

Subsystem: A self-contained system within a larger system.

Network traffic: Network traffic or data traffic is the amount of data moving across a network at a given point of time.

Database: A structured set of data held in a computer, especially one that is accessible in various ways.

Cloud System: Actually, there is no cloud system. There is only someone's computer. This system means you can access this computer anytime anywhere. Your data will be stored in another data storage and through internet you can use that storage area however you like.

Data: Characteristic or information. This means data are a set of values and variables about one or more person or object.

User Interface: The means by which the user and a computer system interact, in particular the use of input devices and software.

Open Source: Open source software is software with source code that anyone can inspect, modify, and enhance.

6 References

1) https://app.diagrams.net/ : We drew our diagrams on this website.
2) https://www.cs.fsu.edu/~myers/cop3331/notes/sysdesign.html : This website helped us to choose the architecture of the system.
3) Object-Oriented Software Engineering Using UML, Patterns, and Java 3rd Edition by Bernd Bruegge & Allen H.Dutoit