



Democratic and Popular Republic of Algeria
Ministry of Higher Education and Scientific Research
University of Mohamed Khider - BISKRA
Faculty of Exact Sciences, Natural and Life Sciences

Department of Computer Science

Order Number: :
1/M1 AI/2024

Mini project report

1rd year Academic Master

Specialty: Artificial Intelligence

File Sharing Client-Server System

Directed by :

- Assassi Salah eddine.
- Nouar El Mouataz Billah.
- Bekkari Abderrahmane
- Azri mohamed
- Meftah Abderrahim

University year: 2024-2025

1. Evolution of file sharing

1.1 Introduction

1.1.1 General summary

File sharing involves the exchange of digital files—such as documents, music, videos and software—between multiple users over networks or the Internet. Common methods include peer-to-peer (P2P) networks, cloud storage services, and file transfer protocols (FTP). It is widely used for collaboration, data sharing, and backups. Initially, file sharing relied on physical media, such as floppy disks and CDs, which were constrained by slow processes and physical limitations. The Internet revolutionized this practice, introducing more efficient methods such as email attachments and FTP servers. The emergence of P2P networks, exemplified by platforms such as Napster, marked a significant shift, enabling direct sharing between users. Although this innovation increased efficiency, it also raised issues around copyright infringement and data security. Today, cloud storage services such as Dropbox, Google Drive, and OneDrive have further advanced file sharing, offering features such as seamless access, automatic backups, and improved collaboration tools.

1.1.2 History of File Sharing

1.1.2.1 The Early Physical Media Era (1970s–1980s)

- 1970s:
 - **File Transfer Protocol (FTP)** was introduced, allowing digital file sharing over early networks like ARPANET.
 - File sharing largely depended on physical media such as magnetic tapes.
- 1980s:
 - **Floppy disks** became the primary medium for sharing files, offering portable but limited storage.
 - **Bulletin Board Systems (BBS)** enabled users to exchange files over dial-up connections, marking an early move to digital file exchange.

1.1.2.2 The Internet Emergence Era (1990s)

- 1990s:
 - The widespread adoption of the Internet introduced file sharing through **email attachments** and early **FTP servers**.
 - Users began sharing larger files, although bandwidth limitations still posed challenges.

1.1.2.3 The Peer-to-Peer (P2P) Revolution (1999–2005)

- 1999:
 - Napster launched as the first popular P2P file-sharing platform, focusing on music file exchange.
 - Napster's decentralized approach allowed users to share files directly, bypassing central servers.
- 2001:
 - Napster was shut down following legal action over copyright issues, but the P2P model inspired successors.
- 2000–2005:
 - Platforms like Kazaa, LimeWire, and BitTorrent emerged, expanding P2P sharing to all types of digital files.
 - BitTorrent (2001) introduced a new protocol for sharing large files efficiently, becoming a cornerstone of P2P technology.

1.1.2.4 The Cloud Storage Era (2006–Present)

- 2006:
 - Dropbox introduced the concept of cloud-based file sharing, offering a secure, centralized platform for sharing and syncing files.
- 2010s:
 - Platforms like Google Drive (2012) and Microsoft OneDrive (2014) entered the scene, making file sharing accessible across devices with advanced collaboration tools.
- 2020s:
 - Cloud services have become the dominant method of file sharing, offering features like real-time collaboration, enhanced security, and global access.
 - Advances in encryption and compliance standards address privacy concerns associated with file sharing.

1.2 File Transfer Protocol

1.2.1 General summary

File Transfer Protocol (FTP) is a standard network protocol developed in 1971 for transferring files between a client and a server over a network, such as the Internet. It operates using a **client-server model**, where a client connects to a remote server to upload, download, or manage files.

FTP uses two separate communication channels:

1. **Control Channel (port 21)**: Handles commands and responses between the client and server.
2. **Data Channel (port 20)**: Used for transferring files.

Users can authenticate with a username and password, although anonymous FTP access is sometimes allowed. It supports both **active** and **passive** modes, providing flexibility for different network configurations. FTP was among the first protocols to enable digital file sharing and is still used in various applications today.

1.2.2 Keywords

- **FTP (File Transfer Protocol)**: A protocol used for transferring files between a client and server over a network. It uses separate channels for commands and data.
- **SFTP (Secure FTP)**: A secure version of FTP that encrypts both the command and data channels for safe file transfers over a network.
- **P2P (Peer-to-Peer)**: A decentralized file-sharing method where users exchange files directly between devices, without needing a central server.
- **Cloud Storage**: Online platforms (e.g., Google Drive, Dropbox) that allow users to store and share files remotely, accessible from any device.
- **FTP Server**: A system that stores files and allows users to upload, download, and manage them via FTP connections.

1.2.3 Purpose of using

- **File Transfer**: Moving files between a client and server, such as uploading or downloading data.
- **Remote File Management**: Managing files on remote servers (renaming, deleting, etc.) without physical access.
- **Batch Transfers**: Transferring multiple files or directories at once, saving time.
- **Automation**: Automating tasks like scheduled backups or software updates.

-
- **Platform Independence:** Enabling file transfers between different operating systems (Windows, Linux, macOS).
 - **Secure Transfers:** Ensuring security with secure FTP variants like **SFTP** or **FTPS**.

1.3 Conclusion

The world is now witnessing its fifth generation, which relies heavily on technology in many areas, the most important of which is: the data transfer aspect, through the use of user interfaces, which are considered the point of interaction between the user and the server, and the most important of which is: because it is the fastest and easiest to reach the user, which led to the creation of many technologies. In the next chapter, we will talk about designing our application.

2. Application design

2.1 Introduction

2.1.1 General summary

This chapter will provide the application design phase, starting with definition and followed by different diagrams types which are: class diagram, use case diagram, sequence diagram.

2.1.2 UML

UML (Unified Modeling Language) is a standardized modeling language used in software engineering and system design. It provides a set of graphical notations to represent various aspects of a system, including its structure, behavior, and interactions. UML diagrams help communicate complex ideas and concepts among stakeholders during the software development process.

- Type of diagrams

1. Use Case Diagram:

- Depicts interactions between **actors** (users or external systems) and **use cases**.
- Shows **functional requirements** from a user's perspective.
- Useful for **requirements analysis** and **system design**.

2. Class Diagram:

- Represents the **static structure** of a system.
- Shows **classes**, their **attributes**, **methods**, and **relationships**.
- Useful for understanding the **class hierarchy** and **inter dependencies**.

3. Sequence Diagram:

- Displays **object interactions over time**.
- Shows the **sequence of messages** exchanged between objects.
- Useful for understanding **dynamic behavior** and **collaborations**.

4. Activity Diagram:

- Visualizes **workflows**, **processes**, and **business logic**.
- Includes **activities**, **control flow**, and **decision points**.
- Useful for modeling **business processes** and **workflow automation**.

2.2 Diagrams

2.2.1 Use Case diagrams

A use case diagram illustrates the functionality provided by the system through several actors, in our case the user.

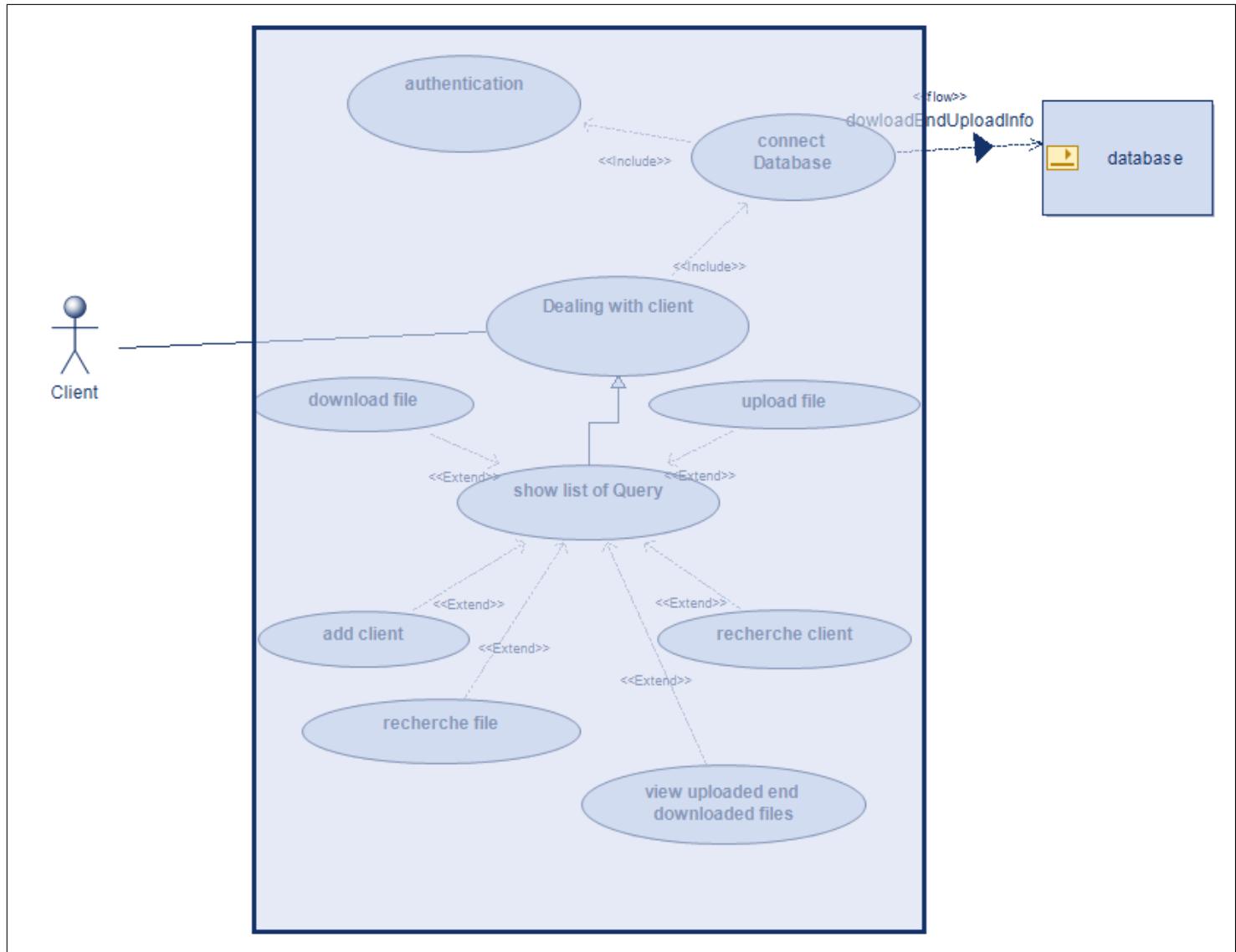


Figure 2.1: use case diagram

2.2.2 Class diagrams

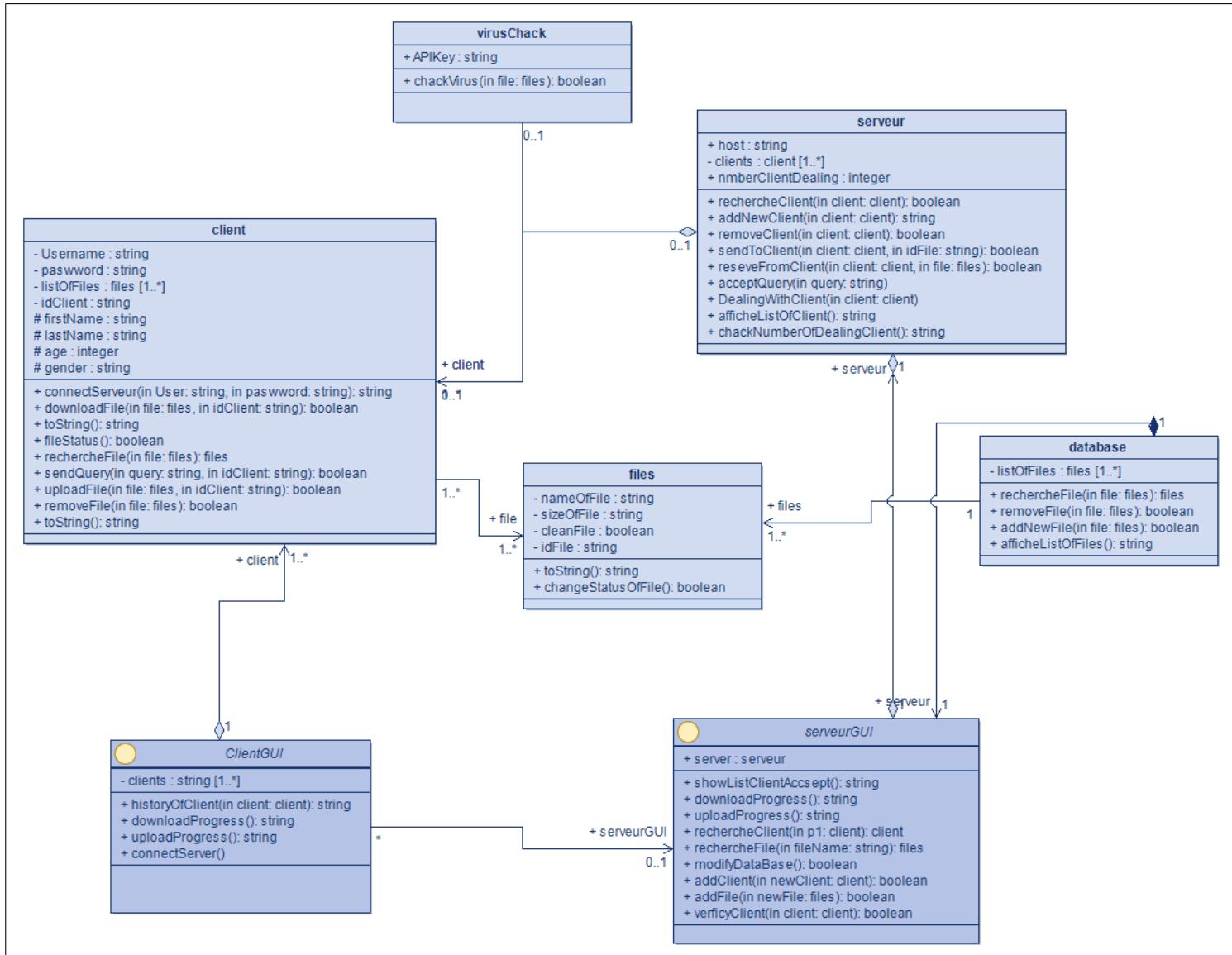


Figure 2.2: class diagram

2.2.3 Sequence diagrams

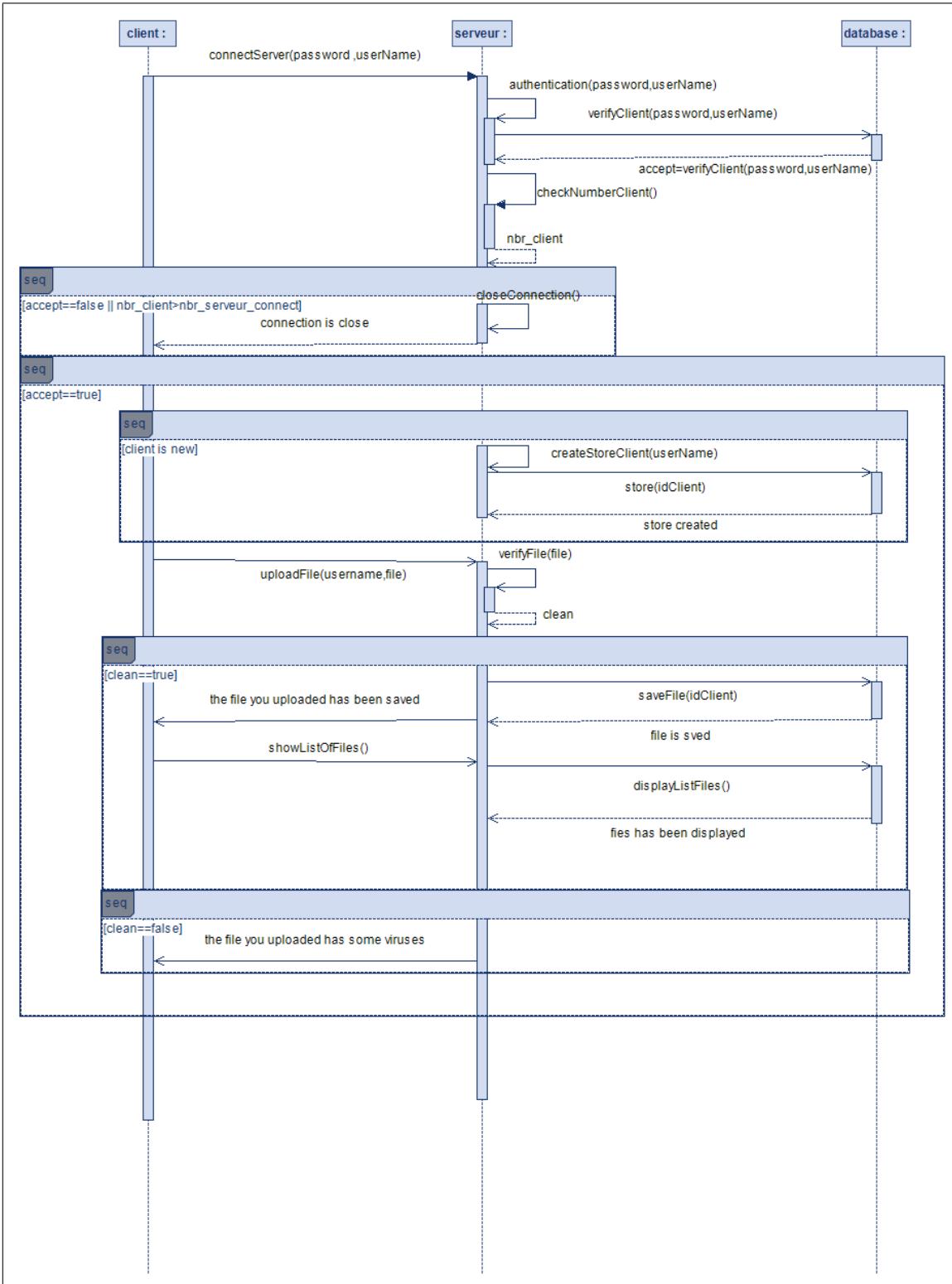


Figure 2.4: Patient sequence diagram

2.3 Conclusion

The focus of this section was to study and create a desktop application to manage file sharing between the client and the server. We created our design consisting of a use case, a class, and a sequence, which will help us determine the languages we need to build the application and its overall appearance.

3. Realization

3.1 Introduction

3.1.1 General summary

A desktop application is a type of software program designed to run on a computer or workstation, independent of a web browser. It operates locally on a device, leveraging the hardware and operating system to perform tasks. Unlike web-based applications, desktop apps are installed directly on the computer and often do not require an internet connection to function.

According to [Ian Sommerville](#) in [Software Engineering](#), desktop applications are “stand-alone software systems that execute on a client machine and provide functionalities directly to the user without relying on network connections.” This makes them highly reliable for performing critical tasks even in offline environments.

In the book [Programming Windows](#) by [Charles Petzold](#), desktop applications are described as “graphical programs that interact with the user through a combination of windows, menus, and controls, providing a rich and responsive user experience.”

An article in the [International Journal of Software Studies](#) defines desktop applications as “software systems that take full advantage of local computing resources, offering better performance and deeper system integration compared to their web counterparts.”

Desktop applications remain popular in fields requiring high-performance computing, such as design, data analysis, and gaming. They are often preferred for their speed, stability, and ability to function independently of internet reliability.

3.1.2 Purpose of using

1. Performance Optimization

Desktop applications utilize local hardware (CPU, RAM, and storage) for faster file transfers and efficient processing compared to web or mobile apps.

2. Offline Functionality

They can operate offline, enabling file sharing over local networks (e.g., LAN) without requiring internet access, unlike web apps.

3. Handling Large Files

Desktop apps bypass the size limits and restrictions often imposed by web browsers or mobile platforms, allowing seamless transfer of large files.

4. Direct File System Integration

They provide deep integration with the operating system, enabling features like drag-and-drop, context menu actions, and real-time access to local directories.

5. Advanced Protocol Support

Desktop apps can implement custom file-sharing protocols such as P2P or FTP, ensuring flexibility and optimized data transmission without browser or mobile platform constraints.

6. Enhanced Security and Privacy

With options for end-to-end encryption and local network sharing, desktop apps avoid relying on third-party servers, ensuring better control over sensitive data.

7. Resource Efficiency

They minimize dependence on network bandwidth by handling most operations locally, reducing latency and improving reliability.

8. Consistency and Compatibility

Desktop apps are not subject to browser compatibility issues or mobile device limitations, ensuring consistent behavior across platforms.

9. Scalable Features

They support advanced functionalities like transfer prioritization, scheduling, or resuming interrupted transfers—ideal for managing large or multiple file transfers.

10. Superior User Experience

With rich, responsive interfaces and native desktop frameworks, desktop applications provide a smoother, more intuitive user experience compared to web or mobile counterparts.

3.2 Languages used for programming

3.2.1 JAVA

Java is a high-level, object-oriented programming language developed by Sun Microsystems (now owned by Oracle) in 1995. It is designed to be platform-independent, allowing developers to write code once and run it anywhere using the Java Virtual Machine (JVM).

In "Effective Java" by Joshua Bloch, Java is described as "a programming language and computing platform that is robust, secure, and portable, enabling developers to build versatile applications ranging from small-scale utilities to large-scale enterprise solutions."

The book "Core Java Volume I" by Cay S. Horstmann highlights Java's versatility, stating: "Java provides a complete set of libraries for building GUI applications, networking, and database connectivity, making it ideal for cross-platform development."

According to "Java: The Complete Reference" by Herbert Schildt, Java's key features include:

1. Simplicity and readability.
2. Object-oriented principles like inheritance and polymorphism.
3. Automatic memory management via garbage collection.
4. Platform Independence (Write Once, Run Anywhere)
5. Robustness and Rich API.

3.3 The Framework used in programming

3.3.1 JAVAFX

JavaFX is a framework for building rich, interactive user interfaces (UIs) for desktop applications. It was introduced by Oracle as a successor to Swing for creating visually appealing and feature-rich GUI applications.

In "JavaFX 8: Introduction by Example" by Carl Dea et al., JavaFX is described as "a modern framework that integrates seamlessly with Java, providing developers with a wide range of tools for building sophisticated and responsive UIs." The book emphasizes its rich set of UI controls, animation support, and capabilities for creating both 2D and 3D graphical content.

As "JavaFX for Dummies" by Doug Lowe explains, JavaFX supports features like FXML(XML-based markup language for UI definition) and Scene Builder, which provide a more intuitive, visual way of creating UIs. These features make JavaFX an accessible and powerful tool for both beginners and experienced developers.

JavaFX also allows for the creation of responsive UIs that can adjust to various screen sizes and resolutions, making it ideal for both small and large-scale applications.

According to "Pro JavaFX 9" by Johan Vos et al., JavaFX integrates smoothly with Java's capabilities, making it an excellent choice for developers looking to create cross-platform applications that are not only functional but also visually appealing.

3.4 security used in programming

3.4.1 Windows Defender

Windows Defender, now called Microsoft Defender Antivirus, is an integrated security solution built into Windows operating systems to protect against malware, viruses, spyware, and other malicious threats. It is designed to provide real-time protection and ensure that users' systems are secure without requiring additional third-party software.

In "Windows 10 For Dummies" by Andy Rathbone, Windows Defender is described as "a built-in antivirus program that continuously scans files and programs for signs of malicious activity." The book highlights how it runs quietly in the background, offering protection without significant impact on system performance.

As "Windows 10: The Missing Manual" by David Pogue explains, Microsoft Defender provides automatic updates to ensure the antivirus signatures are always up-to-date, minimizing the risk of infection from new threats. It also integrates seamlessly with other Windows security features like firewall and network protection.

In "Microsoft Windows Security Essentials" by Robert L. Mancuso, the role of Windows Defender is discussed as a first line of defense against common cyber threats. The book emphasizes that it offers a simple, user-friendly interface and alerts users if any issues arise.

According to "The Complete Guide to Windows 10" by John D. Walker, Microsoft Defender is especially useful for users who need basic protection without the need for complex configurations or external software. It offers a convenient and reliable way to safeguard personal and business devices from digital threats.

3.4.2 VirusTotal

VirusTotal is an online service that analyzes files and URLs to detect viruses, worms, trojans, and other kinds of malware using multiple antivirus engines and tools. It aggregates results from various antivirus programs and provides a detailed report about whether a file or URL is potentially harmful.

In "Malware Analyst's Cookbook and DVD" by Michael Sikorski and Andrew Honig, VirusTotal is explained as a useful tool for malware researchers. It allows them to quickly identify suspicious files by leveraging a large set of antivirus engines to detect even the most elusive threats.

As described in "Practical Malware Analysis" by Michael Sikorski and Andrew Honig, the VirusTotal API allows users to automate queries and integrate VirusTotal's analysis capabilities into their security workflows. The API can be used to programmatically upload files or submit URLs, retrieve reports, and check file reputation.

The VirusTotal API works by enabling users to submit data (such as files or URLs) for analysis. It checks the data against a multitude of antivirus engines, returning a report that indicates whether any engines detect malicious behavior. This process involves:

- API Key Authentication: Users must sign up and obtain an API key.
- Submission: Files or URLs are submitted through POST requests.
- Analysis and Reporting: Results are returned in JSON format, showing the detection rate and detailed information from each engine.

In "Threat Intelligence and Malware Analysis" by S. S. Jadhav, it is mentioned that companies and security professionals use VirusTotal's API to streamline threat detection, automate analysis of incoming files, and maintain an up-to-date record of potential threats, saving valuable time and resources.

3.4.2.1 companies use it

For companies, the VirusTotal API is integral to **continuous monitoring and automated threat intelligence**, enabling security teams to quickly assess and respond to new malware threats without manual intervention. This makes it a crucial tool for improving overall cybersecurity posture.

Large tech companies like Google, Cisco, and IBM use VirusTotal's API to streamline their cybersecurity efforts. Google, for instance, acquired VirusTotal to enhance its own security features and to provide users with real-time protection against emerging threats. Companies like Cisco use the VirusTotal API to detect and monitor malicious files in their network and systems, ensuring quick response times and enhanced protection.

Additionally, CrowdStrike, a leading cybersecurity firm, integrates VirusTotal's capabilities to bolster its malware detection process. By leveraging VirusTotal's vast database of virus signatures, companies can keep their security solutions up-to-date and responsive to new types of malware, minimizing potential damage and improving threat detection. This level of integration helps organizations maintain an efficient, automated defense system against a wide range of security risks.

3.5 DataBase used in programming

3.5.1 SQLite

SQLite is a self-contained, serverless, zero-configuration, transactional SQL database engine. Unlike other relational database management systems (RDBMS), SQLite is embedded directly into applications, making it ideal for lightweight and mobile applications. It stores data in a single file, which simplifies management and deployment.

In "SQLite For Dummies" by Allen G. Taylor, SQLite is described as an "embedded database engine" that offers full relational database capabilities but without the overhead of setting up or maintaining a server. It supports ACID (Atomicity, Consistency, Isolation, Durability) transactions and is known for its simplicity and efficiency.

According to "Using SQLite" by Jay A. Kreibich, SQLite is perfect for applications that need a database but do not require the complexity and overhead of a full-fledged server-based RDBMS. It is particularly useful in environments where disk space and memory are limited, such as embedded systems, mobile applications, and desktop applications.

As "Beginning SQLite" by Gavin M. O'Rourke explains, SQLite is often used for prototyping or in situations where a small database with minimal administration is required. The book emphasizes its ease of use, with no need for separate server processes or complex setup procedures.

In "Practical SQLite" by Siemen G. O'Hara, it is noted that SQLite is widely adopted in mobile development, notably in Android applications, where it provides a lightweight, reliable solution for local data storage. Its small footprint and reliability make it ideal for use cases ranging from small apps to more complex systems where simplicity and performance are essential.

3.5.2 Purposes of Use it

1. **Lightweight Database Solution:** SQLite is ideal for applications that need a lightweight database without the overhead of managing a full-fledged server, making it perfect for mobile apps, embedded systems, and desktop applications .
2. **Serverless Architecture:** As an embedded database, SQLite doesn't require a separate server process, which simplifies deployment and maintenance .
3. **Zero Configuration:** SQLite requires no configuration or installation, making it easy to integrate into projects without extensive setup .
4. **ACID Compliance:** SQLite provides full ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring reliable transactions even in low-resource environments .
5. **Ideal for Prototyping:** SQLite is commonly used in the early stages of application development for rapid prototyping due to its simplicity and quick setup .
6. **Mobile and Embedded Systems:** It is widely used in mobile devices, such as Android apps, for local data storage, where minimizing storage space is crucial.

General Conclusion

Our project presents a desktop application that allows the user to view the files on the server and interact with them by uploading or downloading files. For this reason, in the first chapter we provided an overview of Sharing in terms of its concept and history. The second chapter is devoted to the conceptual aspect of the site and how to implement the use case of the schema, class and sequence.

In the third chapter we started using implementation tools and appropriate languages to integrate content and style with Java and JavaFX to add more realism and deepen the field of front-end management and databases using SQLite, allowing us to deepen our theoretical and practical knowledge in the field of front-end and databases. (Database Management System).

This project was the subject of an interesting experiment that gave us the opportunity to improve our knowledge and skills in developing and designing desktop applications. Therefore, we consider that we have reached our main goal despite some problems in implementing FTP , which is to build an application that meets the needs of the user by providing an information environment through which he can upload and download files safely, which reduces the burden of ensuring the security of files. We plan to improve the application through the actual experience of many users.