LDAP Organisational User Management Console Application

by

Pranil Bhattacharya

Under the Guidance

of

Ms. Tanisha Biswas

ProcessIT Global Pvt. Ltd.

Our LDAP User Management Project is now hosted on GitHub, allowing for easier access, collaboration, and version control. This repository contains all the necessary files, including the source code, documentation, and any additional resources needed to run the project.

Table of Contents

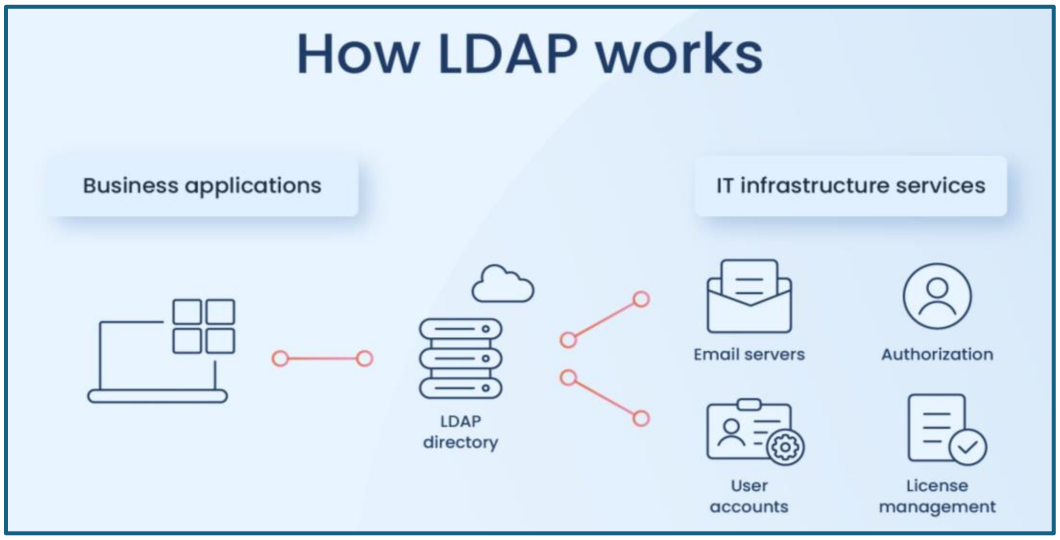
I. Problem Statement

To address the inefficiencies and potential errors in manually managing large-scale user information, we developed a console-based application in C++ that automates the creation, viewing, and deletion of users in an organization's eDirectory from input CSV files.

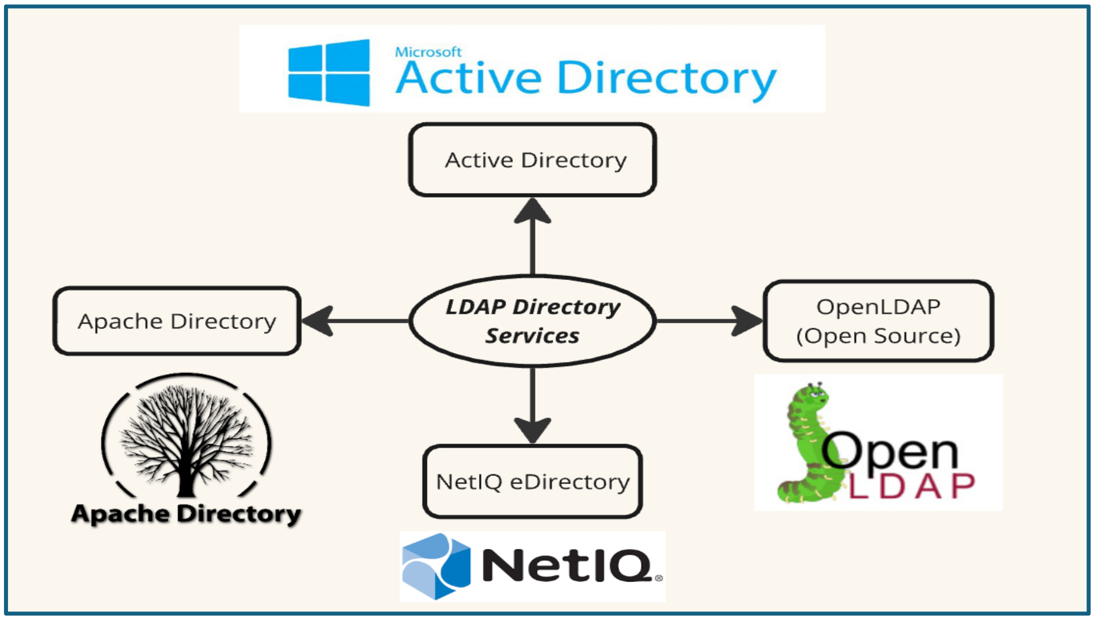
II. Understanding LDAP and its Application

a. Introduction to LDAP

The Lightweight Directory Access Protocol (LDAP) is a protocol used to access and manage directory information over an Internet Protocol (IP) network. LDAP is designed to provide a lightweight, efficient, and standardized way to interact with directory services, which store data about users, groups, devices, and other resources within a network.



2.1 - LDAP working in a nutshell



2.2 – List of popular LDAP directory services. We used NetIQ (Novell) eDirectory

b. Historical Context and Development

LDAP was developed in the early 1990s as a simpler alternative to the more complex Directory Access Protocol (DAP) used by the X.500 directory services. It has since become the de facto standard for directory access, widely adopted in both commercial and open-source directory services.

c. Key Features and Advantages

- ***Lightweight and Efficient***: LDAP is designed to be less resource-intensive compared to other directory access protocols.
- ***Standardized***: Provides a uniform way to access and manage directory information across different platforms and implementations.
- ***Scalable***: Suitable for both small and large-scale deployments, capable of handling millions of entries.
- ***Interoperable***: Compatible with various directory services, including OpenLDAP and eDirectory.

d. Scope and Usage of LDAP

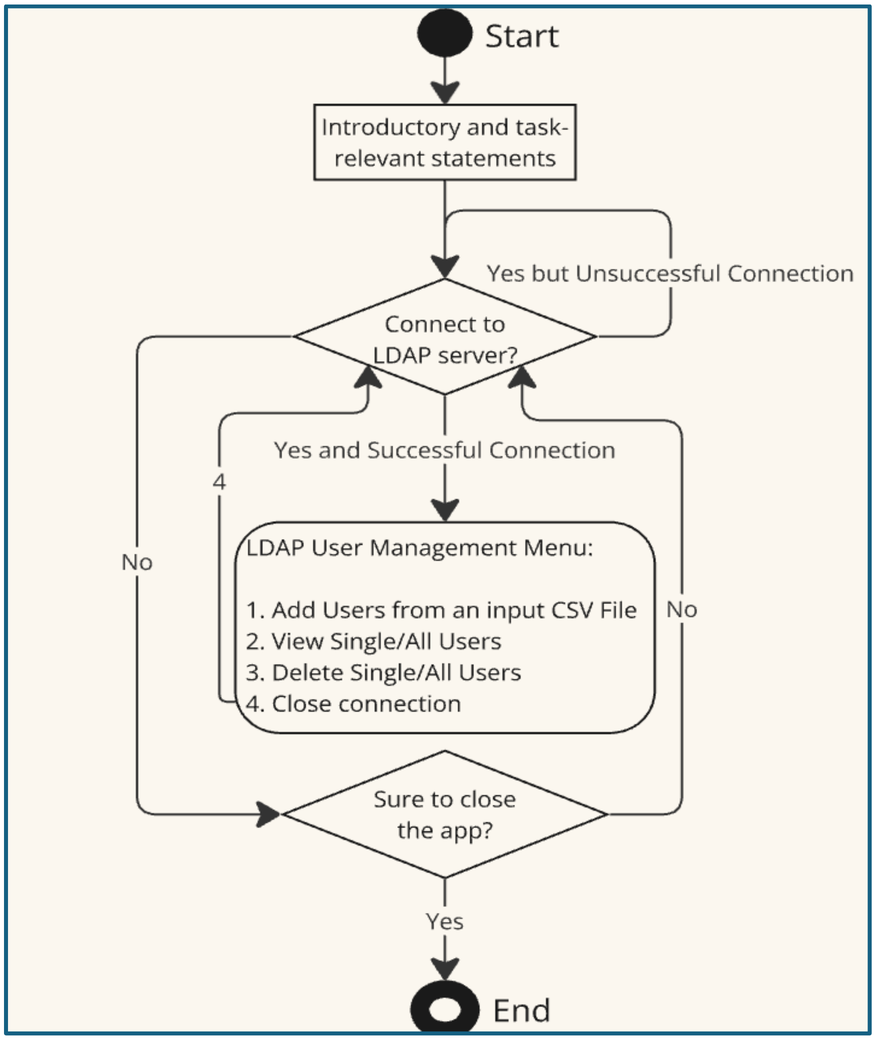
LDAP is commonly used in a variety of applications, including:

- ***Corporate Directories***: Managing employee information, organizational units, and access control.
- ***Email Systems***: Storing and retrieving email addresses and user profiles.
- ***Authentication Services***: Providing centralized authentication for applications and services.
- ***Network Resource Management***: Managing devices, printers, and other network resources.

e. Project-Specific Requirements

For this project, the focus is on developing a console-based application that interacts with an LDAP directory to manage user information. The application will:

1. ***Connect to the LDAP Server***: Bind having administrator details to the specific IP address and port number.
2. ***Read User Data from CSV Files****:* Input user details from CSV files to automate the creation of user accounts.
3. ***Add Users to the LDAP Directory***: Efficiently add new users to the directory based on the provided CSV data.
4. ***View User Information***: Retrieve and display user details from the LDAP directory.
5. ***Delete Users***: Remove user accounts from the directory as needed.

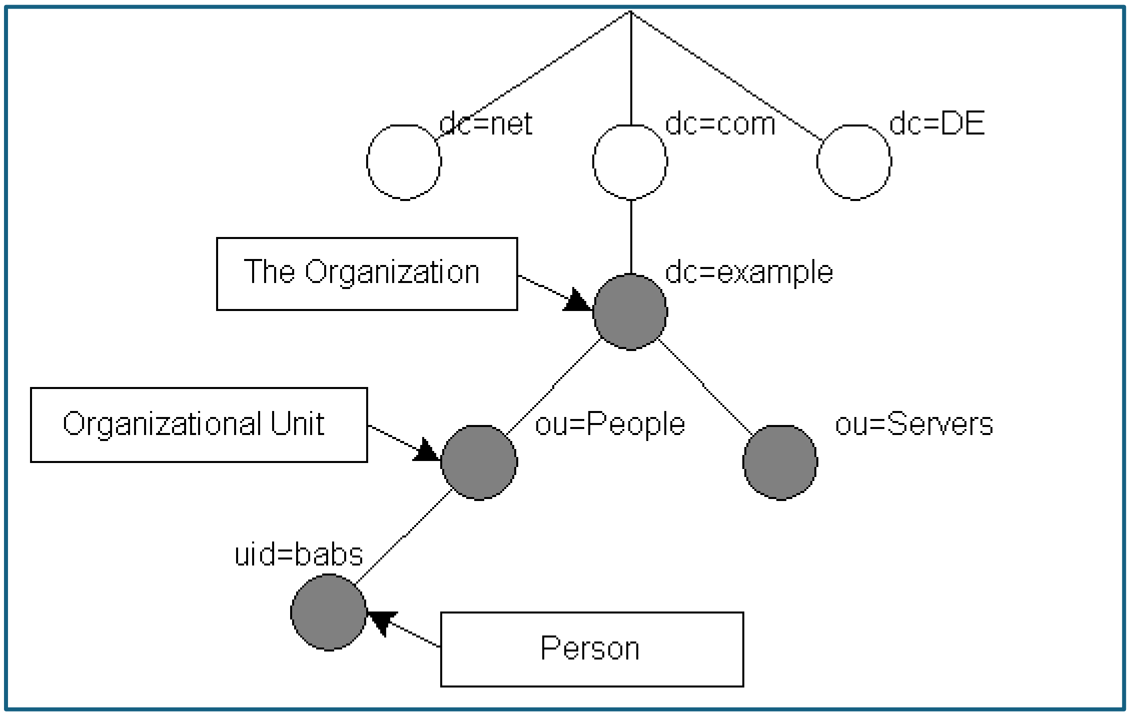

Start
Introductory and task-relevant statements
Yes but Unsuccessful Connection
Connect to LDAP server?
No
Yes and Successful Connection
4
LDAP User Management Menu:
1. Add Users from an input CSV File
2. View Single/All Users
3. Delete Single/All Users
4. Close connection
No
Sure to close the app?
Yes
End

2.3 – A simple flowchart for the application

f. Why LDAP for this Project?

- ***Standardization***: Ensures that the application can work with any LDAP-compliant directory service.
- ***Efficiency***: Automates repetitive tasks, reducing the likelihood of human errors and saving time.
- ***Scalability***: Capable of handling many user entries, making it suitable for organizations of any size.

g. Understanding the LDAP Directory Tree



2.4 - A simple example of the LDAP directory tree

The LDAP directory tree is a hierarchical structure that organizes and stores information in a way that allows for efficient searching and management of data. The structure resembles an inverted tree, with the root at the top and branches extending downwards.

Key Components of the LDAP Directory Tree

1. **Root (Suffix)**
   - The topmost entry in the LDAP directory tree is known as the root or suffix. This entry represents the starting point of the directory hierarchy and is typically denoted by a distinguished name (DN).
   - Example: dc=example,dc=com
2. **Organizational Units (OU)**
   - Organizational Units are containers used to organize entries into a hierarchy within the directory. They help in categorizing and managing groups of related entries.
   - Example: ou=users,dc=example,dc=com
   - In the given image, the "users" organizational unit is a key component under the "c_plusplus_project" directory.
3. **Entries (Leaf Nodes)**
   - Entries are the actual data records stored in the directory. Each entry has a unique distinguished name (DN) and contains a set of attributes and their corresponding values.
   - Example: cn=john.doe,ou=users,dc=example,dc=com
   - In the given image, individual user entries would be added under the "users" organizational unit.
4. **Attributes**
   - Attributes are key-value pairs associated with an entry. Each attribute has a name (attribute type) and one or more values.
   - Example: cn=John Doe, mail=john.doe@example.com
5. **Object Classes**
   - Object classes define the schema for entries, specifying the attributes that an entry can or must have.
   - Example: inetOrgPerson, organizationalPerson, person, top
6. **Hierarchical Structure**

The hierarchical structure of the LDAP directory tree allows for efficient organization and retrieval of data. Each level in the hierarchy represents a different category or grouping, making it easier to manage and search for entries.

- ***Example Tree Structure***:
  - Root: dc=example,dc=com
    - Organizational Unit: ou=users,dc=example,dc=com
      - Entry: cn=john.doe,ou=users,dc=example,dc=com
      - Entry: cn=jane.smith,ou=users,dc=example,dc=com

Visual Representation

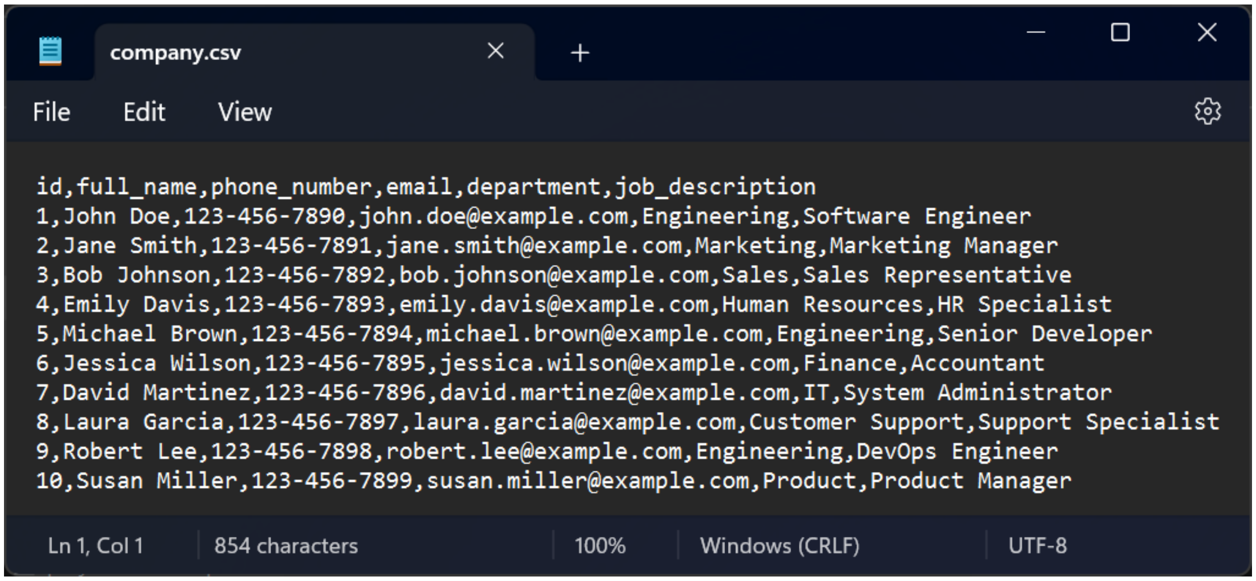
The given image (2.4) visually represents the LDAP directory tree structure. Here's a brief explanation:

- The root of the tree is at the top, with branches extending downwards.
- Each circle represents a node, which can be an organizational unit or an entry.
- The connections between the nodes represent hierarchical relationships.

By organizing data in this hierarchical manner, LDAP allows for scalable and efficient directory services, making it a popular choice for managing user information, access control, and other directory-based services.

h. Understanding the CSV Format

CSV (Comma-Separated Values) is a simple file format used to store tabular data. Each line in a CSV file represents a data record, and each record consists of one or more fields separated by commas. For this project, the CSV file will contain user information like:



```
id,full_name,phone_number,email,department,job_description
1,John Doe,123-456-7890,john.doe@example.com,Engineering,Software Engineer
2,Jane Smith,123-456-7891,jane.smith@example.com,Marketing,Marketing Manager
3,Bob Johnson,123-456-7892,bob.johnson@example.com,Sales,Sales Representative
4,Emily Davis,123-456-7893,emily.davis@example.com,Human Resources,HR Specialist
5,Michael Brown,123-456-7894,michael.brown@example.com,Engineering,Senior Developer
6,Jessica Wilson,123-456-7895,jessica.wilson@example.com,Finance,Accountant
7,David Martinez,123-456-7896,david.martinez@example.com,IT,System Administrator
8,Laura Garcia,123-456-7897,laura.garcia@example.com,Customer Support,Support Specialist
9,Robert Lee,123-456-7898,robert.lee@example.com,Engineering,DevOps Engineer
10,Susan Miller,123-456-7899,susan.miller@example.com,Product,Product Manager
```

2.5 – Our CSV file header and data format example

i. Conclusion

By leveraging LDAP and CSV file handling, this project aims to provide a streamlined and automated solution for managing user information in an organization's directory service, ensuring accuracy, efficiency, and ease of use.

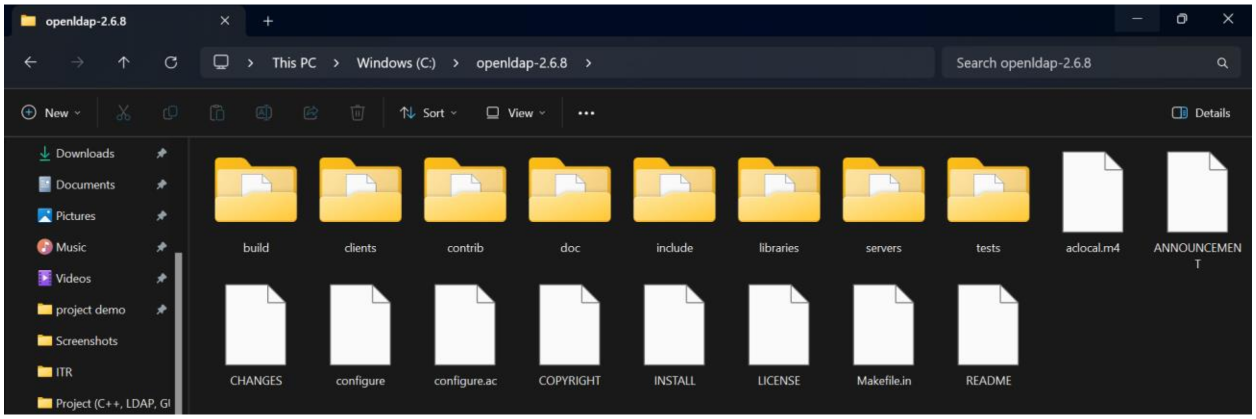
III. Technologies and Tools Used

❖ **C++**

- ***Description***: C++ is a general-purpose programming language known for its performance and efficiency. It supports object-oriented, procedural, and generic programming paradigms.
- ***Why Used***: C++ was chosen for its performance capabilities, especially important for handling potentially large-scale user data in a directory service efficiently.
- ***Key Features***:
    - Object-Oriented Programming: Facilitates organized code and reusability.
    - Standard Library: Provides a wide range of functions and classes for various tasks.
    - Performance: Offers high performance and control over system resources.

❖ **OpenLDAP**

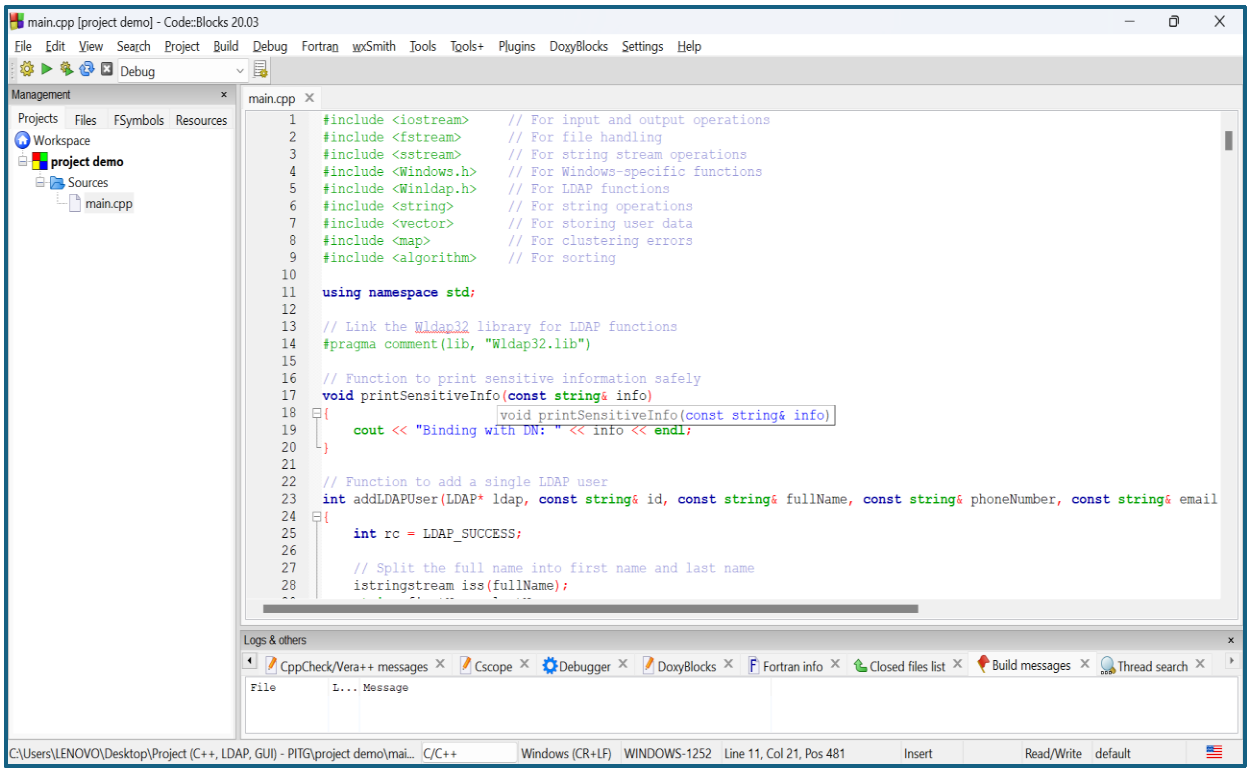
- ***Description***: OpenLDAP is an open-source implementation of the Lightweight Directory Access Protocol (LDAP). It provides a robust and scalable directory service platform.
- ***Why Used***: OpenLDAP was selected for its compliance with LDAP standards, flexibility, and wide adoption in various applications.
- ***Key Features***:
    - *Compatibility*: Works with various directory services and LDAP clients.
    - *Scalability*: Can handle many directory entries.
    - *Extensibility*: Supports various extensions and custom schema definitions.



3.1 – The OpenLDAP (version 2.6.8) Library

❖ **Code::Blocks**

- ***Description:*** Code::Blocks is a free, open-source Integrated Development Environment (IDE) for C, C++, and Fortran. It is designed to be very extensible and fully configurable.
- ***Why Used:*** Code::Blocks was chosen for its user-friendly interface, powerful debugging features, and support for various compilers.
- ***Key Features:***
    - Integrated Debugger: Allows easy debugging of applications.
    - Multiple Compiler Support: Compatible with various compilers like GCC, Clang, and MSVC.
    - Customizable Interface: Users can configure the IDE to suit their development needs.



```
#include <iostream>     // For input and output operations
#include <fstream>      // For file handling
#include <sstream>      // For string stream operations
#include <Windows.h>    // For Windows-specific functions
#include <Winldap.h>    // For LDAP functions
#include <string>       // For string operations
#include <vector>       // For storing user data
#include <map>          // For clustering errors
#include <algorithm>    // For sorting

using namespace std;

// Link the Wldap32 library for LDAP functions
#pragma comment(lib, "Wldap32.lib")

// Function to print sensitive information safely
void printSensitiveInfo(const string& info)
{
    cout << "Binding with DN: " << info << endl;
}

// Function to add a single LDAP user
int addLDAPUser(LDAP* ldap, const string& id, const string& fullName, const string& phoneNumber, const string& email
{
    int rc = LDAP_SUCCESS;

    // Split the full name into first name and last name
    istringstream iss(fullName);
```

main.cpp [project demo] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
Management
Projects Files FSymbols Resources
Workspace
project demo
Sources
main.cpp
Logs & others
CppCheck/Vera++ messages Cscope Debugger DoxyBlocks Fortran info Closed files list Build messages Thread search
File L... Message
C:\Users\LENOVO\Desktop\Project (C++, LDAP, GUI) - PITG\project demo\mai... C/C++ Windows (CR+LF) WINDOWS-1252 Line 11, Col 21, Pos 481 Insert Read/Write default

3.2 – The Code::Blocks IDE

❖ **Wldap32**

- ***Description***: Wldap32.dll is a Windows system file that provides functions to interact with LDAP directories. It is part of the Windows API used for network administration and management.
- ***Why Used***: Wldap32 was utilized to leverage Windows-specific LDAP functions for managing user data in the directory service.
- ***Key Features***:
    - LDAP API Functions: Provides a range of functions for connecting to, querying, and modifying LDAP directories.
    - Integration: Seamlessly integrates with C++ applications on Windows.

IV. Libraries Used

1. <iostream>

- **What is it?**: <iostream> is a part of the C++ Standard Library used for input and output operations.
- **Is it a part of C++, Windows, or other?**: It is part of the C++ Standard Library.
- **How is it configured or linked in the IDE?**: This library is pre-configured in any standard C++ development environment and does not require additional linking or configuration.
- **Why and how is it used?**: Used for reading from the standard input (e.g., cin) and writing to the standard output (e.g., cout).
- **Significance**: Fundamental for console-based input and output operations in C++.

2. <fstream>

- **What is it?**: <fstream> is a part of the C++ Standard Library used for file handling.
- **Is it a part of C++, Windows, or other?**: It is part of the C++ Standard Library.
- **How is it configured or linked in the IDE?**: This library is pre-configured in any standard C++ development environment and does not require additional linking or configuration.
- **Why and how is it used?**: Used to handle file I/O operations such as opening, reading, writing, and closing files.
- **Significance**: Essential for applications that need to persist data or read data from files.

3. <sstream>

- **What is it?**: <sstream> is a part of the C++ Standard Library used for string stream operations.
- **Is it a part of C++, Windows, or other?**: It is part of the C++ Standard Library.
- **How is it configured or linked in the IDE?**: This library is pre-configured in any standard C++ development environment and does not require additional linking or configuration.
- **Why and how is it used?**: Used to perform input and output operations on strings, similar to file streams.
- **Significance**: Useful for parsing and formatting strings.

4. <Windows.h>

- **What is it?**: <Windows.h> is a Windows-specific header file that provides declarations for all of the functions in the Windows API.
- **Is it a part of C++, Windows, or other?**: It is specific to the Windows operating system.
- **How is it configured or linked in the IDE?**: Automatically included with development environments that support Windows application development, such as Visual Studio.
- **Why and how is it used?**: Used to perform Windows-specific operations like handling Windows GUI, system services, and more.
- **Significance**: Crucial for developing applications that interact directly with the Windows operating system.

5. <Winldap.h>

- **What is it?**: <Winldap.h> is a Windows-specific header file for LDAP (Lightweight Directory Access Protocol) functions.
- **Is it a part of C++, Windows, or other?**: It is specific to the Windows operating system.
- **How is it configured or linked in the IDE?**: Must be linked with the Wldap32.lib library using the #pragma comment (lib, "Wldap32.lib") directive or by setting it in the project properties in the IDE.
- **Why and how is it used?**: Used to interact with LDAP directories, performing operations such as adding, deleting, and searching for users.
- **Significance**: Essential for applications that need to manage directory services.

6. <string>

- **What is it?**: <string> is a part of the C++ Standard Library that provides support for string operations.
- **Is it a part of C++, Windows, or other?**: It is part of the C++ Standard Library.
- **How is it configured or linked in the IDE?**: This library is pre-configured in any standard C++ development environment and does not require additional linking or configuration.
- **Why and how is it used?**: Used to manipulate and manage strings in C++.
- **Significance**: Fundamental for handling textual data.

7. <vector>

- **What is it?**: <vector> is a part of the C++ Standard Library that provides a dynamic array, which can resize itself automatically when an element is added or removed.
- **Is it a part of C++, Windows, or other?**: It is part of the C++ Standard Library.
- **How is it configured or linked in the IDE?**: This library is pre-configured in any standard C++ development environment and does not require additional linking or configuration.
- **Why and how is it used?**: Used to store and manage collections of elements that can change size dynamically.
- **Significance**: Essential for managing collections of data efficiently.

8. <map>

- **What is it?**: <map> is a part of the C++ Standard Library that provides an associative array, which stores elements in key-value pairs.
- **Is it a part of C++, Windows, or other?**: It is part of the C++ Standard Library.
- **How is it configured or linked in the IDE?**: This library is pre-configured in any standard C++ development environment and does not require additional linking or configuration.
- **Why and how is it used?**: Used to store elements that are accessed by keys rather than indices.
- **Significance**: Useful for fast retrieval of data based on keys.

9. <algorithm>

- **What is it?**: <algorithm> is a part of the C++ Standard Library that provides a collection of functions to perform various operations like sorting, searching, and manipulating data.
- **Is it a part of C++, Windows, or other?**: It is part of the C++ Standard Library.
- **How is it configured or linked in the IDE?**: This library is pre-configured in any standard C++ development environment and does not require additional linking or configuration.
- **Why and how is it used?**: Used to perform common operations on containers and sequences.
- **Significance**: Provides efficient implementations of common algorithms, saving development time and improving code quality.

V. C++ Structures Used

1. **Namespace (using namespace std;)**
   - **What is it?**: A namespace is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc.) inside it.
   - **Is it a part of C++, Windows, or other?**: It is part of the C++ Standard Library.
   - **How is it configured or linked in the IDE?**: This is a standard part of C++ and does not require special configuration.
   - **Why and how is it used?**: Used to avoid naming conflicts and to group related code into logical structures.
   - **Significance**: Helps in organizing code and preventing name collisions.
2. **Function Definitions**
   - **What is it?**: A function definition provides the actual body of the function.
   - **Is it a part of C++, Windows, or other?**: It is part of the C++ language.
   - **How is it configured or linked in the IDE?**: Functions are defined in the code and do not require special linking.
   - **Why and how is it used?**: Used to encapsulate code into reusable blocks.
   - **Significance**: Essential for modular programming and code reuse.
3. **Class and Object-Oriented Structures**
   - **What is it?**: Classes and objects are fundamental concepts in object-oriented programming, encapsulating data and functions into a single entity.
   - **Is it a part of C++, Windows, or other?**: It is part of the C++ language.
   - **How is it configured or linked in the IDE?**: Classes are defined in the code and do not require special linking.
   - **Why and how is it used?**: Used to model real-world entities and relationships.
   - **Significance**: Crucial for building complex and maintainable software systems.
4. **Standard Template Library (STL) Containers (e.g., vector, map)**
   - **What is it?**: STL containers are data structures provided by the C++ Standard Library.
   - **Is it a part of C++, Windows, or other?**: They are part of the C++ Standard Library.
   - **How is it configured or linked in the IDE?**: These are pre-configured and do not require additional linking.
   - **Why and how is it used?**: Used to store and manage collections of data efficiently.
   - **Significance**: They provide generic, reusable data structures and algorithms, reducing the need for custom implementations.

5. **Preprocessor Directives (#include, #pragma comment)**
   - **What is it?**: Preprocessor directives are commands that are processed by the preprocessor before compilation.
   - **Is it a part of C++, Windows, or other?**: They are part of the C++ language, and some are specific to Windows.
   - **How is it configured or linked in the IDE?**: These are written directly in the code and interpreted by the compiler.
   - **Why and how is it used?**: Used to include header files, define macros, and link libraries.
   - **Significance**: Essential for setting up the environment and managing dependencies.
6. **String Manipulation (e.g., std::string, std::istringstream)**
   - **What is it?**: String manipulation involves operations on string data types, provided by the C++ Standard Library.
   - **Is it a part of C++, Windows, or other?**: It is part of the C++ Standard Library.
   - **How is it configured or linked in the IDE?**: This is pre-configured in any standard C++ development environment.
   - **Why and how is it used?**: Used to process and manipulate text data.
   - **Significance**: Essential for handling and processing string data efficiently.
7. **Error Handling (e.g., cerr)**
   - **What is it?**: Error handling involves mechanisms to manage errors and exceptions during program execution.
   - **Is it a part of C++, Windows, or other?**: It is part of the C++ Standard Library.
   - **How is it configured or linked in the IDE?**: This is pre-configured in any standard C++ development environment.
   - **Why and how is it used?**: Used to output error messages and manage error conditions.
   - **Significance**: Crucial for robust and reliable software development.

VI. Notable Code Snippets

a. Initialization and Set-Up

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <windows.h>
#include <winldap.h>

#pragma comment(lib, "wldap32.lib")

LDAP* ldapConnect()
{
    LDAP* ldap;
    ULONG version = LDAP_VERSION3;

    ldap = ldap_init("ldap.example.com", LDAP_PORT);
    if (ldap == NULL)
    {
        std::cerr << "Failed to initialize LDAP." << std::endl;
        return NULL;
    }

    ldap_set_option(ldap, LDAP_OPT_PROTOCOL_VERSION, &version);

    ULONG result = ldap_simple_bind_s(ldap,
"cn=admin,dc=example,dc=com","password…");
    if (result != LDAP_SUCCESS)
    {
        std::cerr << "Failed to bind to LDAP. Error: " <<
ldap_err2string(result) << std::endl;
        ldap_unbind(ldap);
        return NULL;
    }
    return ldap;
}
```

Explanation

- ***Purpose:*** This code snippet initializes and sets up the LDAP connection.
- ***Include Headers and Libraries:*** Includes necessary headers (<iostream>, <fstream>, <string>, <vector>, <windows.h>, <winldap.h>) and links the wldap32.lib library.
- ***Define Function ldapConnect():*** Initializes the LDAP connection using ldap_init(). If the initialization fails, it logs an error and returns NULL.
- ***Set LDAP Protocol Version:*** Sets the LDAP protocol version to 3 using ldap_set_option().
- ***Bind to LDAP Server:*** Attempts to bind to the LDAP server using ldap_simple_bind_s() with provided credentials ("cn=admin,dc=example,dc=com", "password"). If the bind fails, it logs an error and returns NULL.
- ***Return LDAP Connection Object:*** If successful, returns the LDAP connection object.

b. CSV File Handling and Validation

```
bool validateCSVHeader(const std::string& header)
{
    return header == "uid,cn,sn,mail";
}

std::vector<std::vector<std::string>> readCSV(const std::string& filename)
{
    std::ifstream file(filename);
    std::string line;
    std::vector<std::vector<std::string>> data;

    if (file.is_open())
    {
        getline(file, line);
        if (!validateCSVHeader(line))
        {
            std::cerr << "Invalid CSV header. Please enter the correct
file." << std::endl;
            return {};
        }
        while (getline(file, line))
        {
            std::stringstream ss(line);
            std::string token;
            std::vector<std::string> row;
            while (getline(ss, token, ','))
            {
                row.push_back(token);
            }
            data.push_back(row);
        }
        file.close();
    }
    else
    {
        std::cerr << "Unable to open file." << std::endl;
    }
    return data;

}
```

Explanation

- ***Purpose***: This code snippet handles reading and validating CSV files.
- ***Steps***:
    1. Validate CSV Header: Defines a function validateCSVHeader() to check if the CSV header matches the expected format "uid,cn,sn,mail".
    2. Read CSV File: Defines a function readCSV() to read data from a CSV file. It opens the file using std::ifstream.
    3. Check Header: Reads the first line and validates the header using validateCSVHeader(). If invalid, logs an error and returns an empty vector.
    4. Parse CSV Lines: Reads and parses each line of the CSV file into a vector of strings, splitting by commas using std::stringstream. Stores each row in a 2D vector.
    5. Return Data: Returns the parsed data as a 2D vector of strings.

c. User Management Functions

i. Add Users

```
void addUser(LDAP* ldap, const std::vector<std::string>& userDetails)
{
    LDAPMod mod[4];
    LDAPMod *mods[5];
    std::string dn = "uid=" +userDetails[0] + ",ou=users,dc=example,dc=com";

    char* objectClassVals[] = { "inetOrgPerson", NULL };
    mod[0].mod_op = LDAP_MOD_ADD;
    mod[0].mod_type = "objectClass";
    mod[0].mod_vals.modv_strvals = objectClassVals;

    char* cnVals[] = { const_cast<char*>(userDetails[1].c_str()), NULL };
    mod[1].mod_op = LDAP_MOD_ADD;
    mod[1].mod_type = "cn";
    mod[1].mod_vals.modv_strvals = cnVals;

    char* snVals[] = { const_cast<char*>(userDetails[2].c_str()), NULL };
    mod[2].mod_op = LDAP_MOD_ADD;
    mod[2].mod_type = "sn";
    mod[2].mod_vals.modv_strvals = snVals;

    char* mailVals[] = { const_cast<char*>(userDetails[3].c_str()), NULL };
    mod[3].mod_op = LDAP_MOD_ADD;
    mod[3].mod_type = "mail";
    mod[3].mod_vals.modv_strvals = mailVals;

    mods[0] = &mod[0];
    mods[1] = &mod[1];
    mods[2] = &mod[2];
    mods[3] = &mod[3];
    mods[4] = NULL;

    int result = ldap_add_ext_s(ldap, dn.c_str(), mods, NULL, NULL);
    if (result != LDAP_SUCCESS)
    {
        std::cerr << "Failed to add user " << userDetails[0] << ". Error: " <<
ldap_err2string(result) << std::endl;
    }
    else
    {
        std::cout << "User " << userDetails[0] << " added successfully." <<
std::endl;
    }
}

void addUsers(LDAP* ldap, const std::vector<std::vector<std::string>>& users)
{
    for (const auto& user : users)
    {
        addUser(ldap, user);
    }
}
```

Explanation

- ***Purpose***: This code snippet handles adding users to the LDAP directory.
- ***Steps***:
    1. Define Function addUser(): Constructs the distinguished name (DN) for the user.
    2. Set LDAP Attributes: Sets LDAP attributes (objectClass, cn, sn, mail) using LDAPMod structures.
    3. Add User: Adds the user to the LDAP directory using ldap_add_ext_s(). If the operation fails, logs an error.
    4. Define Function addUsers(): Iterates through the provided user data and calls addUser() for each user.

ii. View Users

```
void viewUsers(LDAP* ldap)
{
    LDAPMessage* result;
    LDAPMessage* entry;
    BerElement* ber;
    char* attribute;
    char** values;

    int resultCode = ldap_search_s(ldap, "ou=users,dc=example,dc=com", 
LDAP_SCOPE_SUBTREE, "(objectClass=inetOrgPerson)", NULL, 0, &result);
    if (resultCode != LDAP_SUCCESS)
    {
        std::cerr << "Failed to search LDAP directory. Error: " << 
ldap_err2string(resultCode) << std::endl;
        return;
    }

    for (entry = ldap_first_entry(ldap, result); entry != NULL; entry = 
ldap_next_entry(ldap, entry))
    {
        std::cout << "DN: " << ldap_get_dn(ldap, entry) << std::endl;

        for (attribute = ldap_first_attribute(ldap, entry, &ber); attribute 
!= NULL; attribute = ldap_next_attribute(ldap, entry, ber))
        {
            values = ldap_get_values(ldap, entry, attribute);
            for (int i = 0; values[i] != NULL; i++)
            {
                std::cout << attribute << ": " << values[i] << std::endl;
            }
            ldap_value_free(values);
            ldap_memfree(attribute);
        }

        if (ber != NULL)
        {
            ber_free(ber, 0);
        }

        std::cout << std::endl;
    }
    ldap_msgfree(result);

}
```

Explanation

- ***Purpose***: This code snippet retrieves and displays user details from the LDAP directory.
- ***Steps***:
    1. Define Function viewUsers(): Searches for user entries with objectClass=inetOrgPerson using ldap_search_s().
    2. Iterate Through Results: Uses ldap_first_entry() and ldap_next_entry() to iterate through the search results.
    3. Display User Attributes: For each entry, retrieves and displays the DN and attributes (cn, sn, mail) using ldap_first_attribute(), ldap_next_attribute(), and ldap_get_values(). Attributes and values are printed and memory is freed.
    4. Clean Up: Frees memory allocated for attributes, values, and the LDAP message result using ldap_value_free(), ldap_memfree(), and ldap_msgfree().

iii. Delete Users

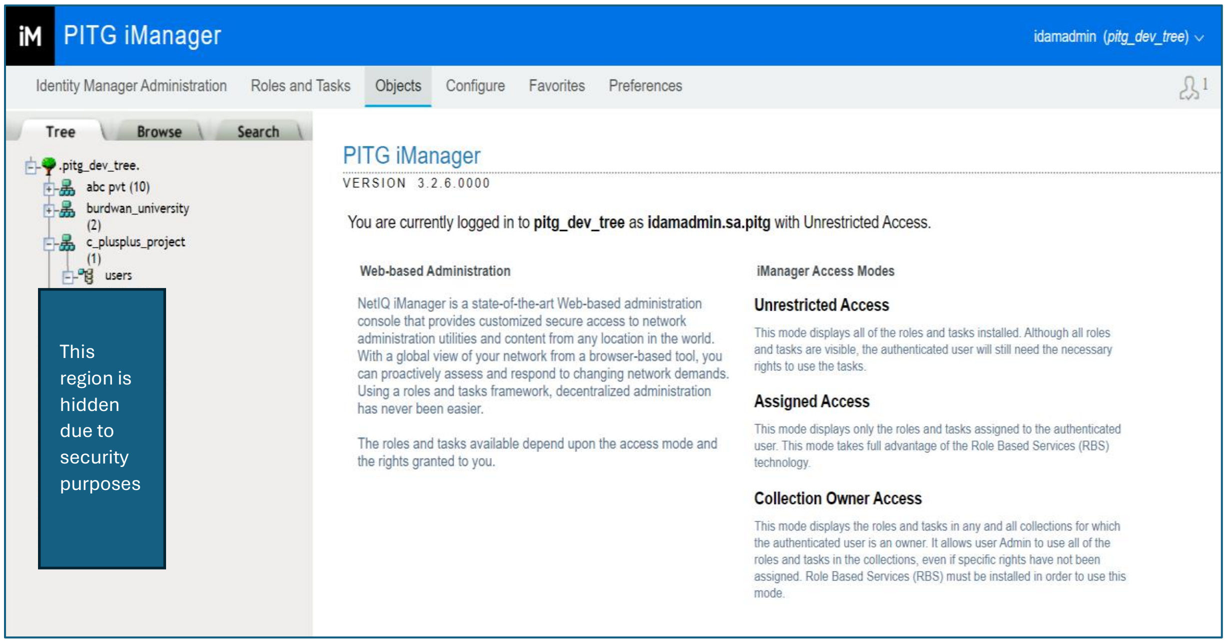
```
void deleteUser(LDAP* ldap, const std::string& uid)
{
    std::string dn = "uid=" + uid + ",ou=users,dc=example,dc=com";
    int result = ldap_delete_s(ldap, dn.c_str());
    if (result != LDAP_SUCCESS)
    {
        std::cerr << "Failed to delete user " << uid << ". Error: " <<
ldap_err2string(result) << std::endl;
    }
    else
    {
        std::cout << "User " << uid << " deleted successfully." <<
std::endl;
    }
}

void deleteUsers(LDAP* ldap, const std::vector<std::string>& uids)
{
    for (const auto& uid : uids)
    {
    deleteUser(ldap, uid);
    }
}
```

Explanation

- ***Purpose:*** This code snippet handles deleting users from the LDAP directory.
- ***Steps:***
  1. Define Function deleteUser(): Constructs the DN for the user using the provided user ID.
  2. Delete User: The ldap_delete_s() function deletes the user from the LDAP directory. If the operation fails, an error message is printed.
  3. Define Function deleteUsers(): Iterates through the provided user IDs and calls deleteUser() for each user.

VII. Test Cases and Expected Outputs

- ***Objective:*** This section mainly documents the test cases for adding, viewing, and deleting LDAP users, along with the expected outputs. But first, we need to connect to the LDAP server.
- ***Tools Used***:
    - **Console Application:** Used to execute the LDAP operations and view the console output.
    - **PITG iManager Web Interface:** Used to verify and manage LDAP users via the "idamadmin" account.



PITG iManager

VERSION 3.2.6.0000

You are currently logged in to **pitg_dev_tree** as **idamadmin.sa.pitg** with Unrestricted Access.

Web-based Administration

NetIQ iManager is a state-of-the-art Web-based administration console that provides customized secure access to network administration utilities and content from any location in the world. With a global view of your network from a browser-based tool, you can proactively assess and respond to changing network demands. Using a roles and tasks framework, decentralized administration has never been easier.

The roles and tasks available depend upon the access mode and the rights granted to you.

iManager Access Modes

Unrestricted Access

This mode displays all of the roles and tasks installed. Although all roles and tasks are visible, the authenticated user will still need the necessary rights to use the tasks.

Assigned Access

This mode displays only the roles and tasks assigned to the authenticated user. This mode takes full advantage of the Role Based Services (RBS) technology.

Collection Owner Access

This mode displays the roles and tasks in any and all collections for which the authenticated user is an owner. It allows user Admin to use all of the roles and tasks in the collections, even if specific rights have not been assigned. Role Based Services (RBS) must be installed in order to use this mode.

5.1 – The PITG iManager: where information is stored hierarchically under a tree as per LDAP protocol.

We are adding Users under "o – c_plusplus_project, ou – users"

1. System is not connected to a Network and/or VPN

Explanation:

In this test case, we aim to demonstrate how the LDAP User Management Application responds when the system is not connected to a network or VPN. This scenario is critical as it tests the application's ability to handle network-related errors gracefully and provide appropriate feedback to the user.

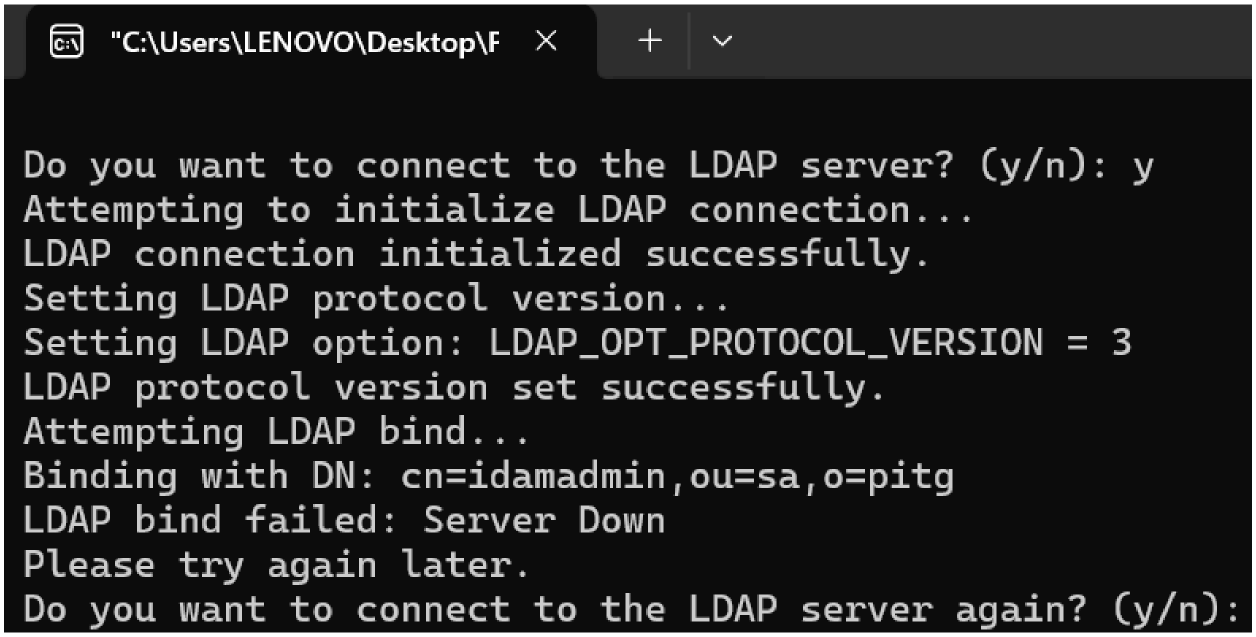
Steps and Observations:

1. Application Introduction:
   - The application displays a welcome message, informing the user of its capabilities, including adding, viewing, and deleting LDAP users.
2. User Prompt for LDAP Connection:
   - The application prompts the user with the question: "Do you want to connect to the LDAP server? (y/n)". In this instance, the user chooses to connect by entering 'y'.
3. Initializing LDAP Connection:
   - The application attempts to initialize the LDAP connection. This step involves setting up the necessary parameters to establish a connection with the LDAP server.
4. Successful Initialization:
   - The application successfully initializes the LDAP connection, as indicated by the message: "LDAP connection initialized successfully."
5. Setting LDAP Protocol Version:
   - The application sets the LDAP protocol version to 3, ensuring compatibility with the server. The success of this operation is confirmed by the message: "LDAP protocol version set successfully."
6. Attempting LDAP Bind:
   - The application attempts to bind to the LDAP server using the provided Distinguished Name (DN) and credentials: "cn=idadmin,ou=sa,o=pitg".
7. Bind Failure Due to Server Down:
   - The bind attempt fails with the message: "LDAP bind failed: Server Down." This failure indicates that the server is unreachable, likely due to the system not being connected to a network or VPN.
8. User Prompt for Retry:
   - The application informs the user to try again later and prompts: "Do you want to connect to the LDAP server again? (y/n)". This allows the user to attempt reconnecting if the network issue is resolved.

Why This Process:

- Network Dependency: LDAP operations require network connectivity to communicate with the LDAP server. This test case highlights the importance of ensuring the system is connected to a network or VPN.
- Error Handling: The application provides clear feedback when it cannot connect to the server, guiding the user to address the network issue before retrying.
- User Experience: By prompting the user to retry, the application enhances user experience by allowing immediate corrective action without restarting the application.

This test case demonstrates the robustness of the LDAP User Management Application in handling network-related issues and providing clear, actionable feedback to the user.



```
Do you want to connect to the LDAP server? (y/n): y
Attempting to initialize LDAP connection...
LDAP connection initialized successfully.
Setting LDAP protocol version...
Setting LDAP option: LDAP_OPT_PROTOCOL_VERSION = 3
LDAP protocol version set successfully.
Attempting LDAP bind...
Binding with DN: cn=idamadmin,ou=sa,o=pitg
LDAP bind failed: Server Down
Please try again later.
Do you want to connect to the LDAP server again? (y/n):
```

2. Successful LDAP Connection and User Management Menu Display

Explanation:

In this test case, we demonstrate the process of successfully connecting to the LDAP server and displaying the LDAP User Management Menu. This scenario tests the application's ability to establish a connection with the LDAP server and present the user with available management options.

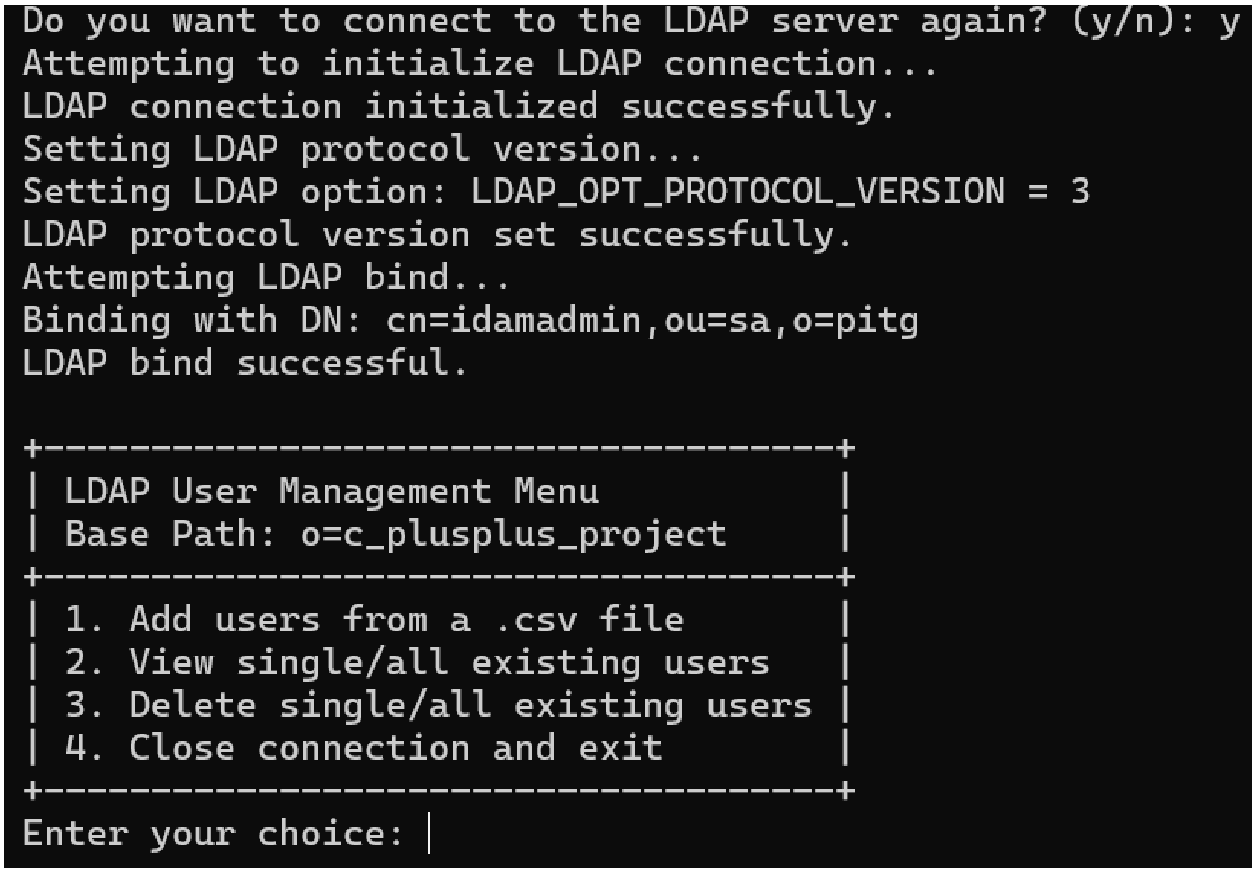
Steps and Observations:

1. User Prompt for LDAP Connection:
    - The application prompts the user with the question: "Do you want to connect to the LDAP server? (y/n)". In this instance, the user chooses to connect by entering 'y'.
2. Initializing LDAP Connection:
    - The application attempts to initialize the LDAP connection. This step involves setting up the necessary parameters to establish a connection with the LDAP server.
3. Successful Initialization:
    - The application successfully initializes the LDAP connection, as indicated by the message: "LDAP connection initialized successfully."
4. Setting LDAP Protocol Version:
    - The application sets the LDAP protocol version to 3, ensuring compatibility with the server. The success of this operation is confirmed by the message: "LDAP protocol version set successfully."
5. Attempting LDAP Bind:
    - The application attempts to bind to the LDAP server using the provided Distinguished Name (DN) and credentials: "cn=idadmin,ou=sa,o=pitg".
6. Successful Bind:
    - The bind attempt is successful, as indicated by the message: "LDAP bind successful."
7. Displaying LDAP User Management Menu:
    - Upon successful connection and bind, the application displays the LDAP User Management Menu with the following options:
        1. Add users from a .csv file
        2. View single/all existing users
        3. Delete single/all existing users
        4. Close connection and exit
8. User Prompt for Menu Selection:
    - The application prompts the user to select an option from the menu by entering the corresponding number.

Why This Process:

- Successful Connection: Establishing a successful connection to the LDAP server is essential for performing any user management operations.
- User Feedback: Providing clear feedback on the connection status ensures the user is informed of the application's progress.
- Menu Display: Displaying the LDAP User Management Menu presents the available operations to the user in an organized manner, enhancing user experience and interaction with the application.
- Robust Operation: The application demonstrates its robustness by successfully navigating through initialization, protocol setting, and binding processes before presenting the management options.

This test case showcases the application's capability to establish a successful connection to the LDAP server and present the user with a clear and functional menu for managing LDAP users.



```
Do you want to connect to the LDAP server again? (y/n): y
Attempting to initialize LDAP connection...
LDAP connection initialized successfully.
Setting LDAP protocol version...
Setting LDAP option: LDAP_OPT_PROTOCOL_VERSION = 3
LDAP protocol version set successfully.
Attempting LDAP bind...
Binding with DN: cn=idamadmin,ou=sa,o=pitg
LDAP bind successful.

+--------------------------------------+
| LDAP User Management Menu            |
| Base Path: o=c_plusplus_project      |
+--------------------------------------+
| 1. Add users from a .csv file        |
| 2. View single/all existing users    |
| 3. Delete single/all existing users  |
| 4. Close connection and exit         |
+--------------------------------------+
Enter your choice:
```

3. Adding Users from a CSV File with Error Handling

Explanation:

In this test case, we demonstrate the process of adding users from a CSV file to the LDAP server, including error handling for incorrect file paths and invalid file formats. This scenario tests the application's ability to validate input and guides the user to provide correct file information.

Steps and Observations:

1. **Displaying LDAP User Management Menu**:
   - The application displays the LDAP User Management Menu with the following options:
     1. Add users from a .csv file
     2. View single/all existing users
     3. Delete single/all existing users
     4. Close connection and exit
2. **User Choice for Adding Users**:
   - The user selects option 1 to add users from a CSV file.
3. **First File Path Input**:
   - The application prompts the user to enter the full path to the CSV file (e.g., C:\path\to\file\company.csv).
   - The user enters an incorrect file path: abc.txt.
4. **Error Handling for Non-existent File**:
   - The application detects that the file does not exist and prompts the user to enter the correct file again with the error message: "Error: The file does not exist. Please enter the correct file again."
5. **Second File Path Input**:
   - The user enters another incorrect file path with an invalid file format: LDAP_User_Management_Flowchart.pdf.
6. **Error Handling for Invalid File Format**:
   - The application detects that the file is not a CSV file and prompts the user to enter the correct file again with the error message: "Error: The file is not a CSV file. Please enter the correct file again."
7. **Third File Path Input**:
   - The user enters the correct file path: company.csv.
8. **Successful User Addition**:
   - The application processes the CSV file and successfully adds all users, as indicated by the message: "All users successfully added: 1 2 3 4 5 6 7 8 9 10."

9. **Returning to LDAP User Management Menu**:
   - The application returns to the LDAP User Management Menu, ready for the next user action.

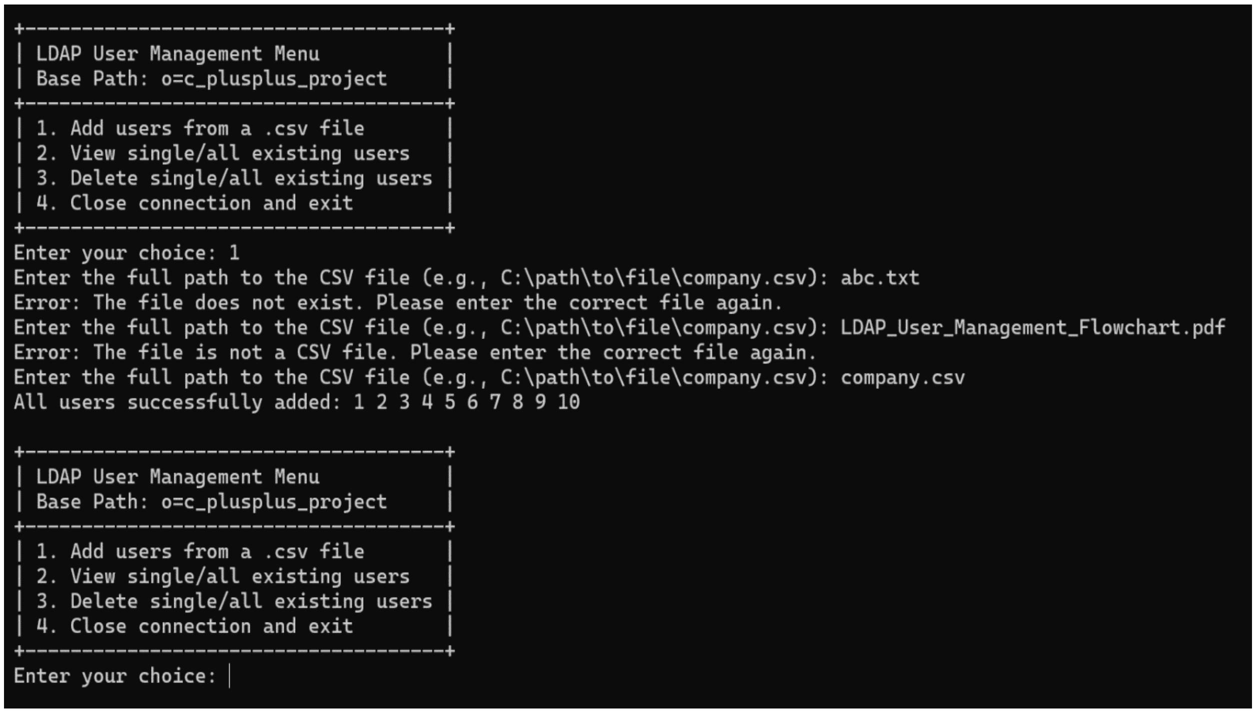
Why This Process:

- **Input Validation**: Ensures that the user provides a valid and existing CSV file path, preventing errors during processing.
- **Error Handling**: Provides clear feedback and guidance to the user when incorrect input is detected, enhancing user experience.
- **Successful Operation**: Demonstrates the application's ability to successfully process valid input and perform the desired operation of adding users.

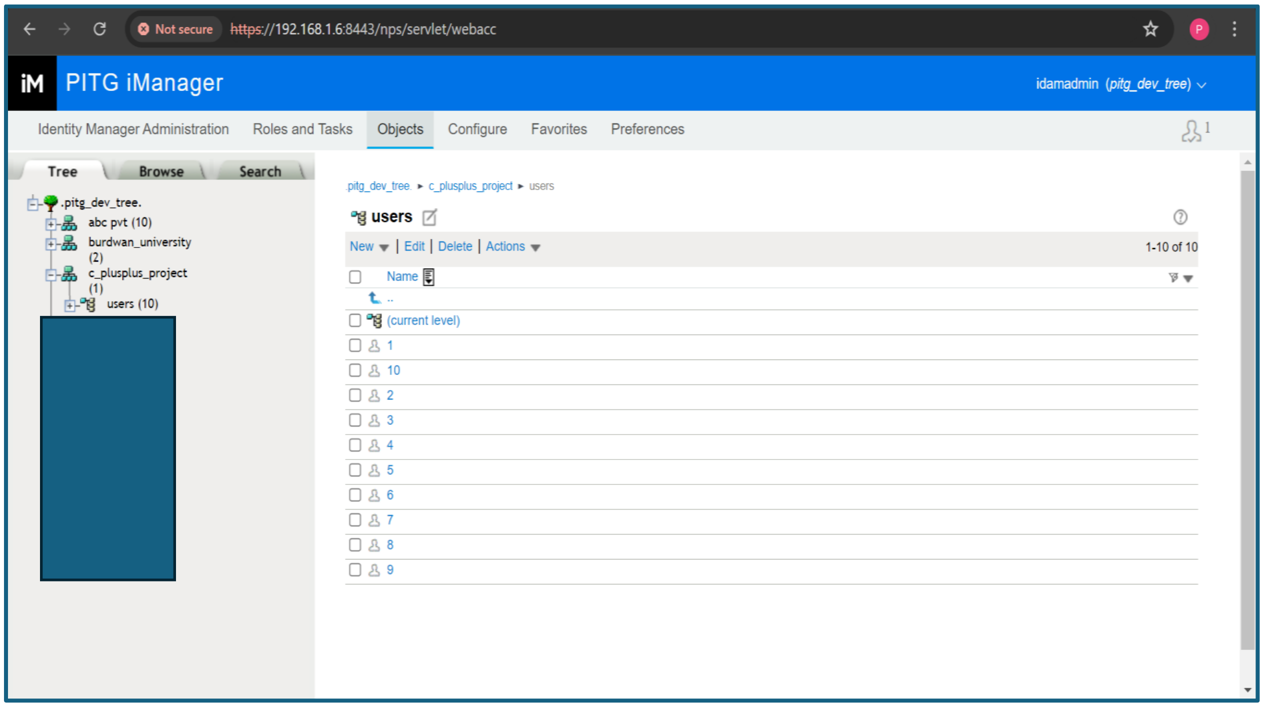
Additional Files:

1. **Directory Structure**:
   - The directory structure shows the project's organization, including the location of CSV files and other relevant documents.
2. **Example CSV File**:
   - The company.csv file contains sample user data, including headers id, full_name, phone_number, email, department, and job_description. This file is used in the test case to add users to the LDAP server.

This test case highlights the application's robustness in handling user input errors and successfully adding users from a valid CSV file.



```
+--------------------------------------+
| LDAP User Management Menu            |
| Base Path: o=c_plusplus_project      |
+--------------------------------------+
| 1. Add users from a .csv file        |
| 2. View single/all existing users    |
| 3. Delete single/all existing users  |
| 4. Close connection and exit         |
+--------------------------------------+
Enter your choice: 1
Enter the full path to the CSV file (e.g., C:\path\to\file\company.csv): abc.txt
Error: The file does not exist. Please enter the correct file again.
Enter the full path to the CSV file (e.g., C:\path\to\file\company.csv): LDAP_User_Management_Flowchart.pdf
Error: The file is not a CSV file. Please enter the correct file again.
Enter the full path to the CSV file (e.g., C:\path\to\file\company.csv): company.csv
All users successfully added: 1 2 3 4 5 6 7 8 9 10

+--------------------------------------+
| LDAP User Management Menu            |
| Base Path: o=c_plusplus_project      |
+--------------------------------------+
| 1. Add users from a .csv file        |
| 2. View single/all existing users    |
| 3. Delete single/all existing users  |
| 4. Close connection and exit         |
+--------------------------------------+
Enter your choice:
```



Not secure https://192.168.1.6:8443/nps/servlet/webacc

iM PITG iManager idamadmin (pitg_dev_tree)

Identity Manager Administration | Roles and Tasks | Objects | Configure | Favorites | Preferences

Tree | Browse | Search

.pitg_dev_tree.
- abc pvt (10)
- burdwan_university (2)
- c_plusplus_project (1)
  - users (10)

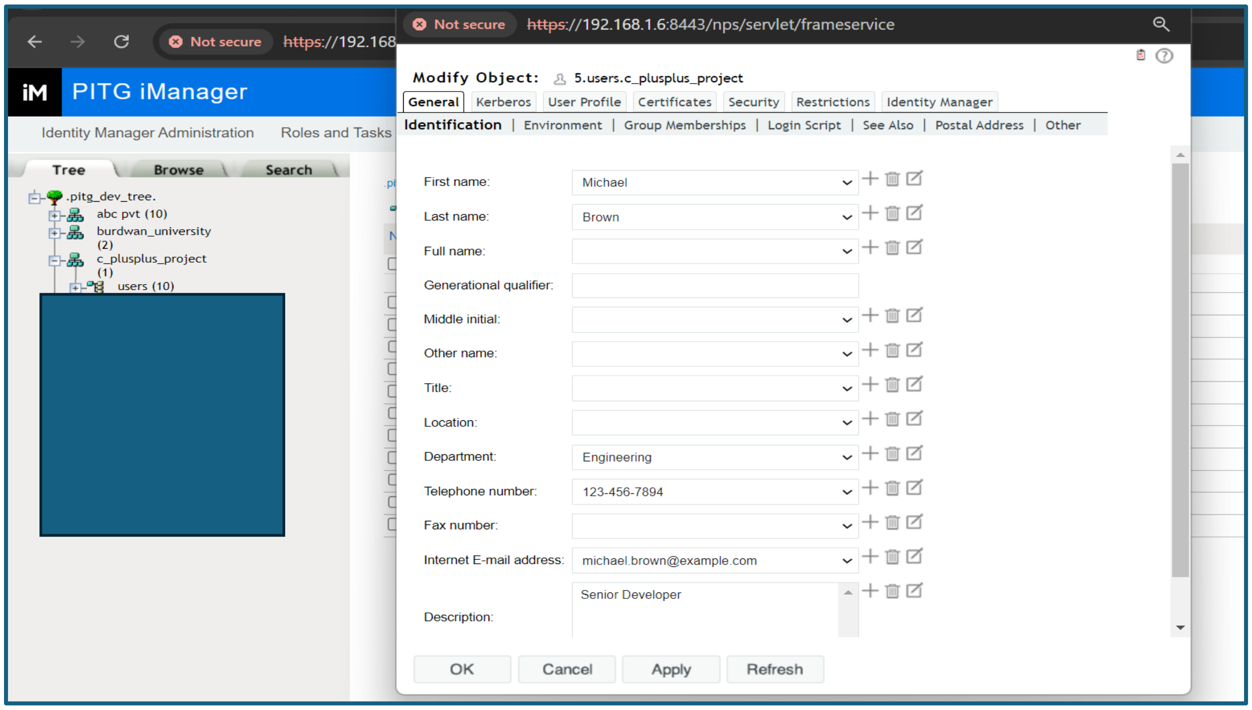
[redacted]

pitg_dev_tree ▸ c_plusplus_project ▸ users

users

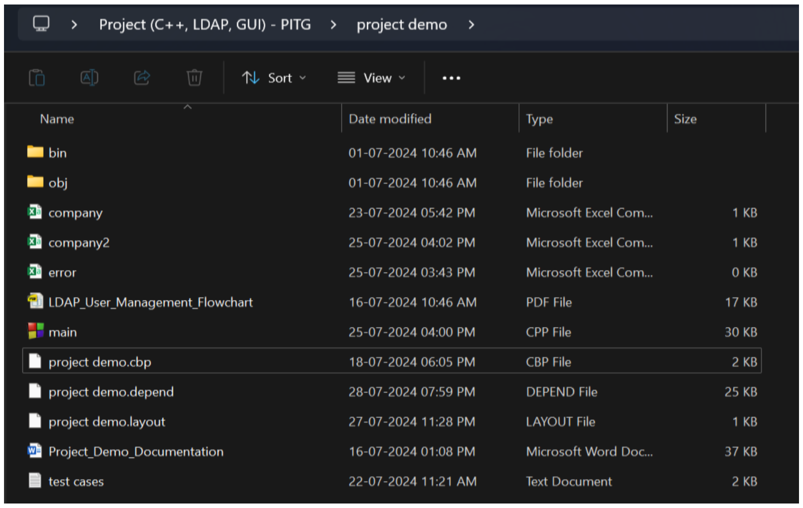
New | Edit | Delete | Actions — 1-10 of 10

Name
..
(current level)
1
10
2
3
4
5
6
7
8
9

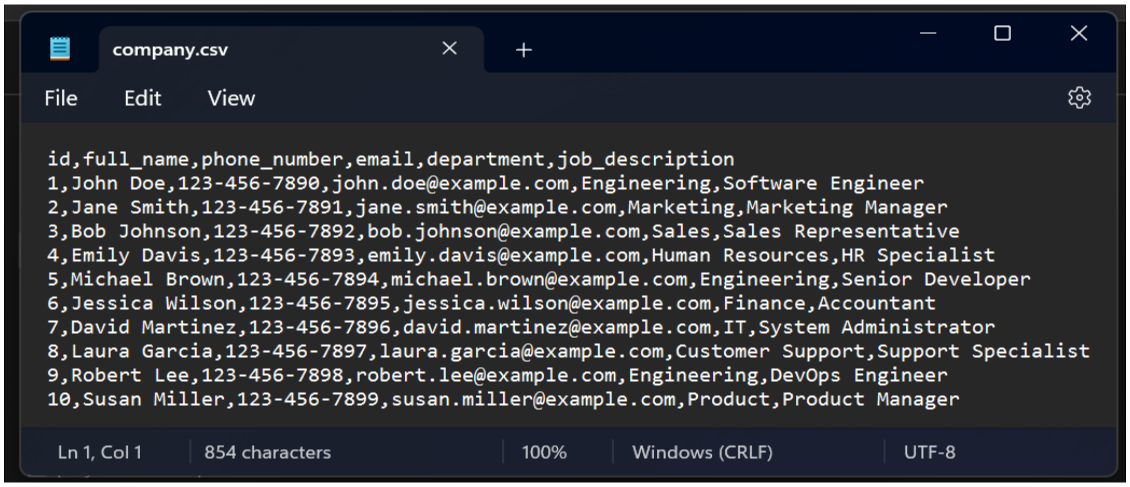


Modify Object: 5.users.c_plusplus_project

First name:	Michael
Last name:	Brown
Full name:	
Generational qualifier:	
Middle initial:	
Other name:	
Title:	
Location:	
Department:	Engineering
Telephone number:	123-456-7894
Fax number:	
Internet E-mail address:	michael.brown@example.com
Description:	Senior Developer



Name	Date modified	Type	Size
bin	01-07-2024 10:46 AM	File folder	
obj	01-07-2024 10:46 AM	File folder	
company	23-07-2024 05:42 PM	Microsoft Excel Com...	1 KB
company2	25-07-2024 04:02 PM	Microsoft Excel Com...	1 KB
error	25-07-2024 03:43 PM	Microsoft Excel Com...	0 KB
LDAP_User_Management_Flowchart	16-07-2024 10:46 AM	PDF File	17 KB
main	25-07-2024 04:00 PM	CPP File	30 KB
project demo.cbp	18-07-2024 06:05 PM	CBP File	2 KB
project demo.depend	28-07-2024 07:59 PM	DEPEND File	25 KB
project demo.layout	27-07-2024 11:28 PM	LAYOUT File	1 KB
Project_Demo_Documentation	16-07-2024 01:08 PM	Microsoft Word Doc...	37 KB
test cases	22-07-2024 11:21 AM	Text Document	2 KB



```
id,full_name,phone_number,email,department,job_description
1,John Doe,123-456-7890,john.doe@example.com,Engineering,Software Engineer
2,Jane Smith,123-456-7891,jane.smith@example.com,Marketing,Marketing Manager
3,Bob Johnson,123-456-7892,bob.johnson@example.com,Sales,Sales Representative
4,Emily Davis,123-456-7893,emily.davis@example.com,Human Resources,HR Specialist
5,Michael Brown,123-456-7894,michael.brown@example.com,Engineering,Senior Developer
6,Jessica Wilson,123-456-7895,jessica.wilson@example.com,Finance,Accountant
7,David Martinez,123-456-7896,david.martinez@example.com,IT,System Administrator
8,Laura Garcia,123-456-7897,laura.garcia@example.com,Customer Support,Support Specialist
9,Robert Lee,123-456-7898,robert.lee@example.com,Engineering,DevOps Engineer
10,Susan Miller,123-456-7899,susan.miller@example.com,Product,Product Manager
```

4. Handling CSV File with No Valid Data Rows

Explanation:

In this test case, we demonstrate the process of handling a scenario where the user attempts to add users from a CSV file that does not contain any valid data rows. This scenario tests the application's ability to validate the content of the CSV file and provide appropriate feedback to the user.

Steps and Observations:

1. **Displaying LDAP User Management Menu**:
    - The application displays the LDAP User Management Menu with the following options:
        1. Add users from a .csv file
        2. View single/all existing users
        3. Delete single/all existing users
        4. Close connection and exit
2. **User Choice for Adding Users**:
    - The user selects option 1 to add users from a CSV file.
3. **File Path Input**:
    - The application prompts the user to enter the full path to the CSV file (e.g., C:\path\to\file\company.csv).
    - The user enters the file path: error.csv.
4. **Error Handling for CSV with No Valid Data Rows**:
    - The application detects that the CSV file does not contain any valid data rows and displays the error message: "Error: CSV file does not contain any valid data rows. Returning to menu."
5. **Returning to LDAP User Management Menu**:
    - The application returns to the LDAP User Management Menu, ready for the next user action.

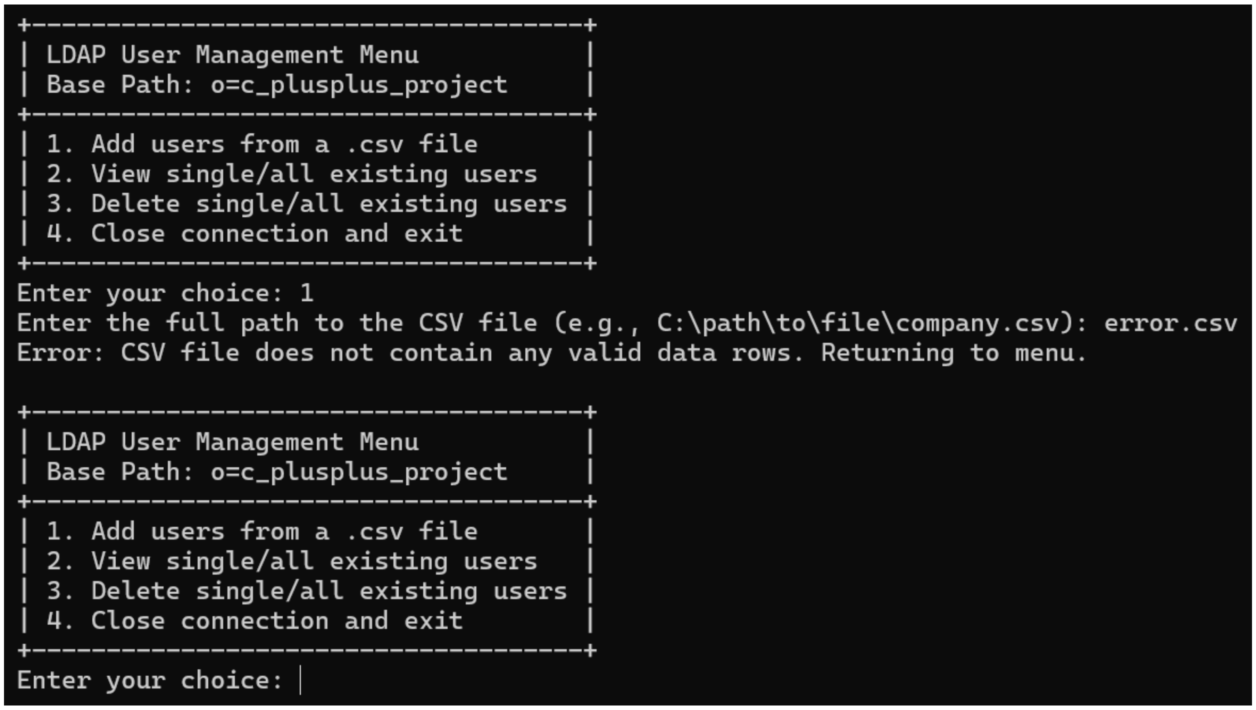
Why This Process:

- **Content Validation**: Ensures that the CSV file contains valid data rows before processing, preventing errors during user addition.
- **Error Handling**: Provides clear feedback and guidance to the user when the CSV file is invalid, enhancing user experience.
- **Robust Operation**: Demonstrates the application's ability to handle invalid input gracefully and return to a functional state.

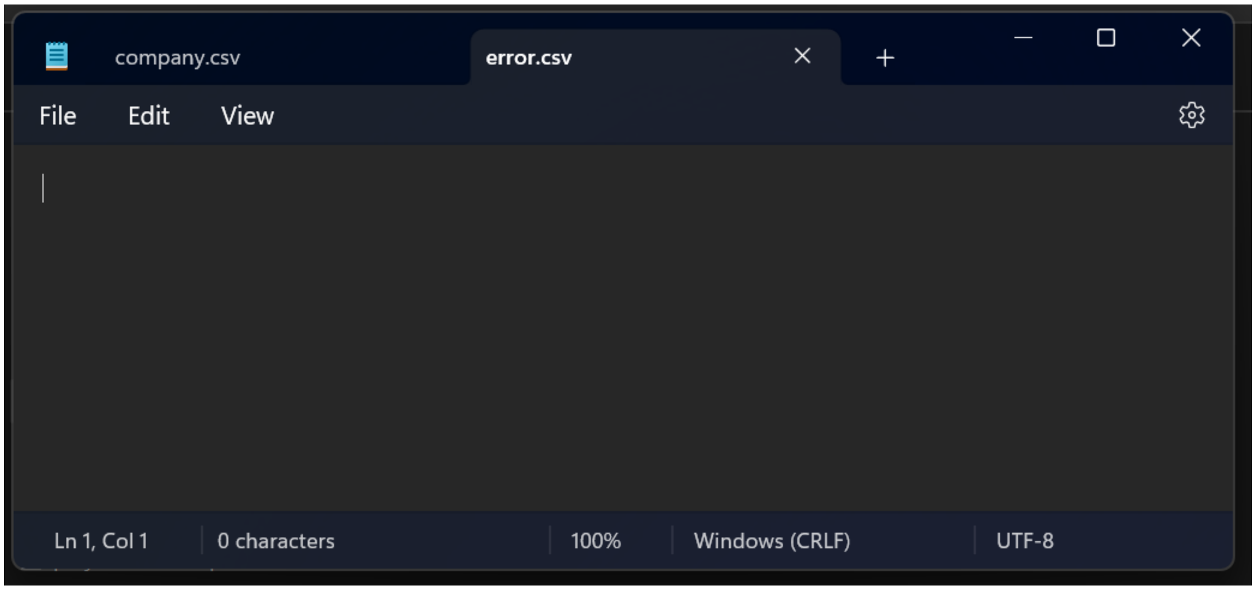
Additional Files:

1. **Empty CSV File**:
    - The error.csv file is shown to be empty, containing no valid data rows. This file is used in the test case to demonstrate the application's validation process.

This test case highlights the application's robustness in handling CSV files with no valid data rows and providing clear feedback to the user.



```
+-------------------------------------+
| LDAP User Management Menu           |
| Base Path: o=c_plusplus_project     |
+-------------------------------------+
| 1. Add users from a .csv file       |
| 2. View single/all existing users   |
| 3. Delete single/all existing users |
| 4. Close connection and exit        |
+-------------------------------------+
Enter your choice: 1
Enter the full path to the CSV file (e.g., C:\path\to\file\company.csv): error.csv
Error: CSV file does not contain any valid data rows. Returning to menu.

+-------------------------------------+
| LDAP User Management Menu           |
| Base Path: o=c_plusplus_project     |
+-------------------------------------+
| 1. Add users from a .csv file       |
| 2. View single/all existing users   |
| 3. Delete single/all existing users |
| 4. Close connection and exit        |
+-------------------------------------+
Enter your choice:
```



5. Viewing LDAP Users

Explanation:

In this test case, we demonstrate the process of viewing LDAP users, including handling scenarios where the specified user does not exist. This scenario tests the application's ability to retrieve and display user information from the LDAP server accurately.

Steps and Observations:

1. **Displaying LDAP User Management Menu**:
    - The application displays the LDAP User Management Menu with the following options:
        1. Add users from a .csv file
        2. View single/all existing users
        3. Delete single/all existing users
        4. Close connection and exit
2. **User Choice for Viewing Users**:
    - The user selects option 2 to view users.
3. **Single User Search (Non-existent User)**:
    - The application prompts the user to specify whether to view a single user or all users (single/all). The user chooses 'single'.
    - The application then prompts the user to enter the user ID (cn). The user enters an invalid ID: abc.
    - The application performs an LDAP search and returns the message: "LDAP search failed: No Such Object", indicating that the user does not exist.
4. **Returning to LDAP User Management Menu**:
    - The application returns to the LDAP User Management Menu, ready for the next user action.
5. **Single User Search (Existing User)**:
    - The user selects option 2 to view users.
    - The user chooses 'single' and enters a valid user ID: 3.
    - The application performs an LDAP search and retrieves the details for the user with ID 3, displaying the information:
        - **cn**: 3
        - **sn**: Johnson
        - **givenName**: Bob
        - **mail**: bob.johnson@example.com

        - **ou**: Sales
        - **telephoneNumber**: 123-456-7892
        - **description**: Sales Representative

6. **Returning to LDAP User Management Menu**:
    - The application returns to the LDAP User Management Menu, ready for the next user action.
7. **Viewing All Users**:
    - The user selects option 2 to view users.
    - The user chooses 'all'.
    - The application retrieves and displays the details of all existing users in the LDAP directory, listing each user's information in a clear format.

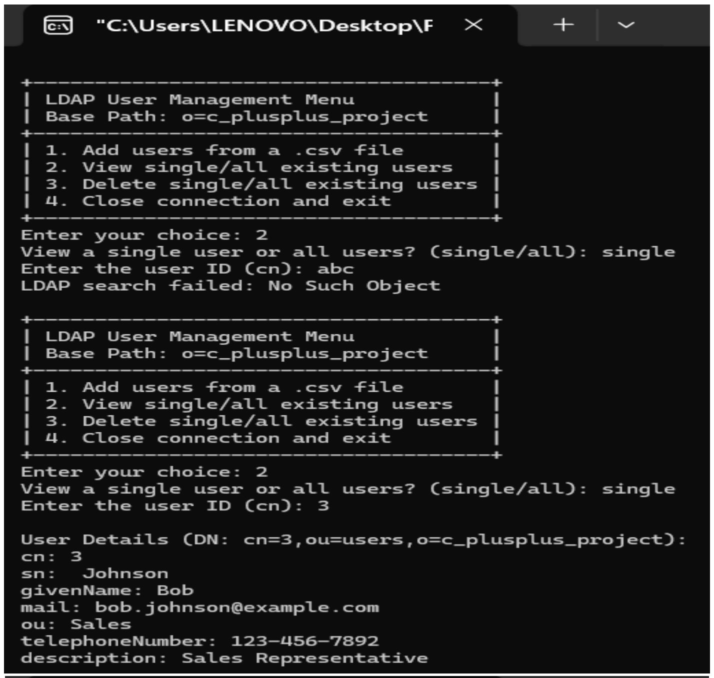
Why This Process:

- **User Retrieval**: Demonstrates the application's ability to retrieve user information from the LDAP server based on specified criteria (single or all users).
- **Error Handling**: Provides clear feedback when the specified user does not exist, guiding the user to correct their input.
- **Comprehensive Display**: Ensures that user details are displayed in a structured and readable format, enhancing user experience.

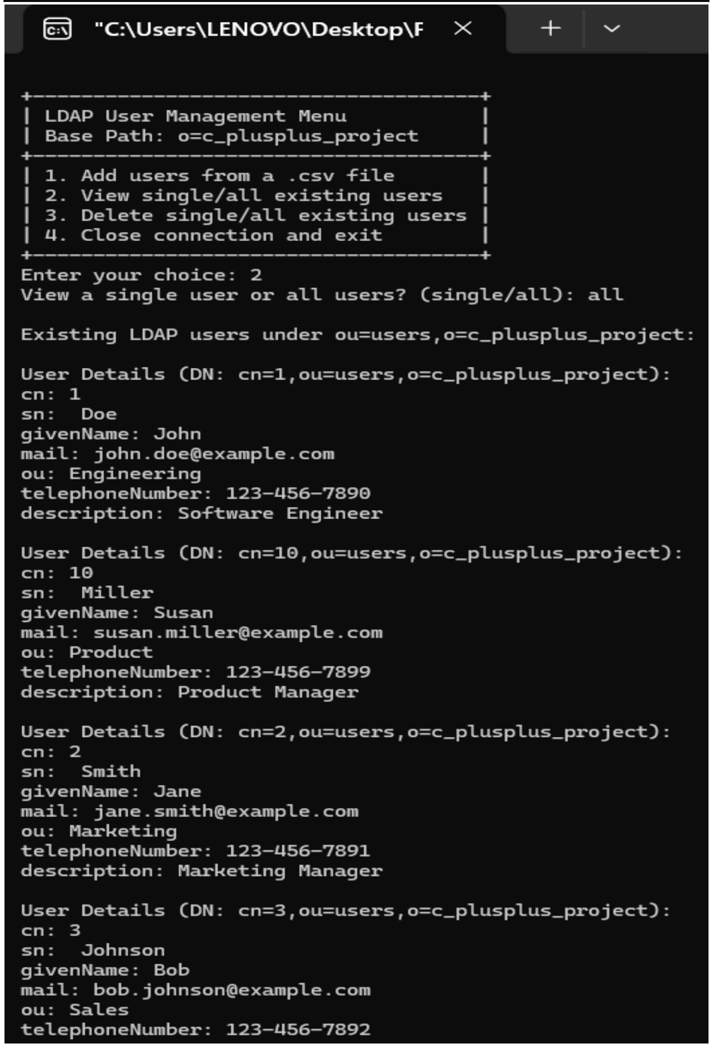
Additional Files:

1. **Non-existent User Search**:
    - Shows the error message when the application fails to find a user with the specified ID.
2. **Single User Search**:
    - Displays the details of a specific user retrieved from the LDAP server.
3. **Viewing All Users**:
    - Lists the details of all users in the LDAP directory, providing a comprehensive view of the directory's content.

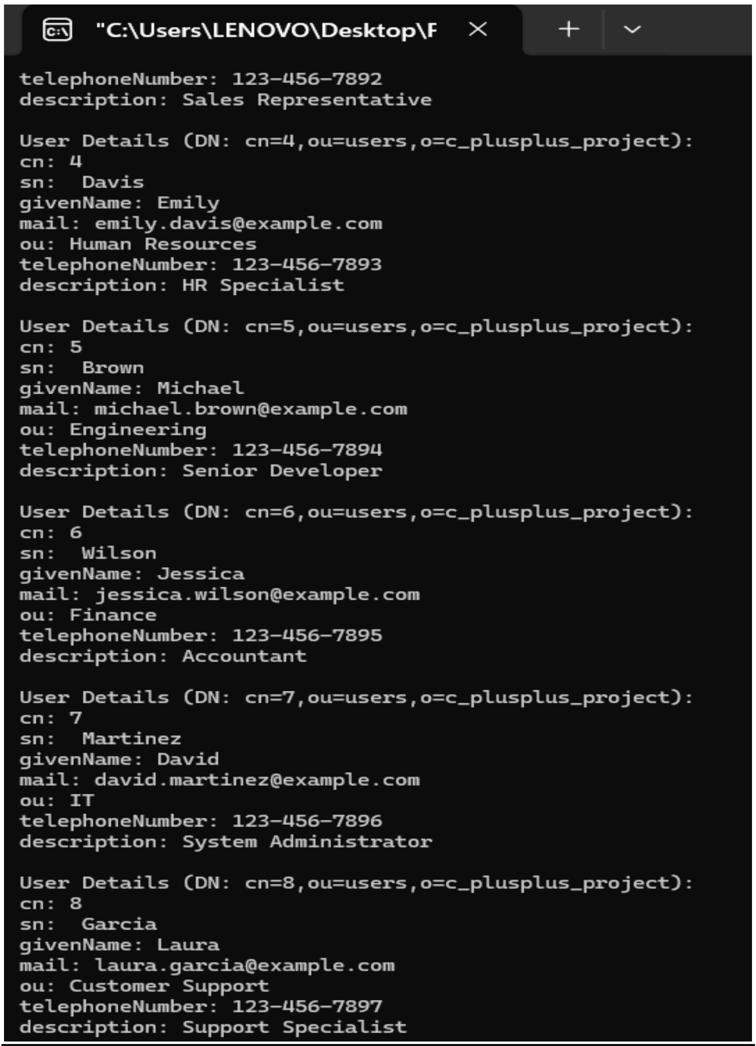
This test case highlights the application's robustness in handling user searches, providing accurate information, and guiding the user in case of errors.



```
"C:\Users\LENOVO\Desktop\F

+--------------------------------------+
| LDAP User Management Menu            |
| Base Path: o=c_plusplus_project      |
+--------------------------------------+
| 1. Add users from a .csv file        |
| 2. View single/all existing users    |
| 3. Delete single/all existing users  |
| 4. Close connection and exit         |
+--------------------------------------+
Enter your choice: 2
View a single user or all users? (single/all): single
Enter the user ID (cn): abc
LDAP search failed: No Such Object

+--------------------------------------+
| LDAP User Management Menu            |
| Base Path: o=c_plusplus_project      |
+--------------------------------------+
| 1. Add users from a .csv file        |
| 2. View single/all existing users    |
| 3. Delete single/all existing users  |
| 4. Close connection and exit         |
+--------------------------------------+
Enter your choice: 2
View a single user or all users? (single/all): single
Enter the user ID (cn): 3

User Details (DN: cn=3,ou=users,o=c_plusplus_project):
cn: 3
sn:  Johnson
givenName: Bob
mail: bob.johnson@example.com
ou: Sales
telephoneNumber: 123-456-7892
description: Sales Representative
```

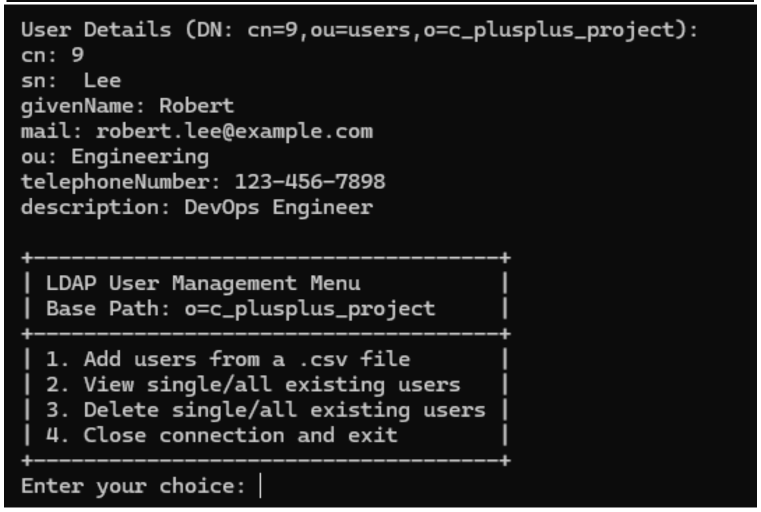


```
"C:\Users\LENOVO\Desktop\F

+--------------------------------------+
| LDAP User Management Menu            |
| Base Path: o=c_plusplus_project      |
+--------------------------------------+
| 1. Add users from a .csv file        |
| 2. View single/all existing users    |
| 3. Delete single/all existing users  |
| 4. Close connection and exit         |
+--------------------------------------+
Enter your choice: 2
View a single user or all users? (single/all): all

Existing LDAP users under ou=users,o=c_plusplus_project:

User Details (DN: cn=1,ou=users,o=c_plusplus_project):
cn: 1
sn:  Doe
givenName: John
mail: john.doe@example.com
ou: Engineering
telephoneNumber: 123-456-7890
description: Software Engineer

User Details (DN: cn=10,ou=users,o=c_plusplus_project):
cn: 10
sn:  Miller
givenName: Susan
mail: susan.miller@example.com
ou: Product
telephoneNumber: 123-456-7899
description: Product Manager

User Details (DN: cn=2,ou=users,o=c_plusplus_project):
cn: 2
sn:  Smith
givenName: Jane
mail: jane.smith@example.com
ou: Marketing
telephoneNumber: 123-456-7891
description: Marketing Manager

User Details (DN: cn=3,ou=users,o=c_plusplus_project):
cn: 3
sn:  Johnson
givenName: Bob
mail: bob.johnson@example.com
ou: Sales
telephoneNumber: 123-456-7892
```



"C:\Users\LENOVO\Desktop\F

```
telephoneNumber: 123-456-7892
description: Sales Representative

User Details (DN: cn=4,ou=users,o=c_plusplus_project):
cn: 4
sn:  Davis
givenName: Emily
mail: emily.davis@example.com
ou: Human Resources
telephoneNumber: 123-456-7893
description: HR Specialist

User Details (DN: cn=5,ou=users,o=c_plusplus_project):
cn: 5
sn:  Brown
givenName: Michael
mail: michael.brown@example.com
ou: Engineering
telephoneNumber: 123-456-7894
description: Senior Developer

User Details (DN: cn=6,ou=users,o=c_plusplus_project):
cn: 6
sn:  Wilson
givenName: Jessica
mail: jessica.wilson@example.com
ou: Finance
telephoneNumber: 123-456-7895
description: Accountant

User Details (DN: cn=7,ou=users,o=c_plusplus_project):
cn: 7
sn:  Martinez
givenName: David
mail: david.martinez@example.com
ou: IT
telephoneNumber: 123-456-7896
description: System Administrator

User Details (DN: cn=8,ou=users,o=c_plusplus_project):
cn: 8
sn:  Garcia
givenName: Laura
mail: laura.garcia@example.com
ou: Customer Support
telephoneNumber: 123-456-7897
description: Support Specialist
```



```
User Details (DN: cn=9,ou=users,o=c_plusplus_project):
cn: 9
sn:  Lee
givenName: Robert
mail: robert.lee@example.com
ou: Engineering
telephoneNumber: 123-456-7898
description: DevOps Engineer

+-------------------------------------+
| LDAP User Management Menu           |
| Base Path: o=c_plusplus_project     |
+-------------------------------------+
| 1. Add users from a .csv file       |
| 2. View single/all existing users   |
| 3. Delete single/all existing users |
| 4. Close connection and exit        |
+-------------------------------------+
Enter your choice: |
```

6. Deleting LDAP Users

Explanation:

In this test case, we demonstrate the process of deleting LDAP users, both single and multiple, from the LDAP directory. This scenario tests the application's ability to remove user entries accurately and reflect the changes in the LDAP directory.

Steps and Observations:

1. **Displaying LDAP User Management Menu**:
    - The application displays the LDAP User Management Menu with the following options:
        1. Add users from a .csv file
        2. View single/all existing users
        3. Delete single/all existing users
        4. Close connection and exit
2. **User Choice for Deleting Users**:
    - The user selects option 3 to delete users.
3. **Single User Deletion**:
    - The application prompts the user to specify whether to delete a single user or all users (single/all). The user chooses 'single'.
    - The application then prompts the user to enter the user ID (cn). The user enters the ID: 2.
    - The application successfully deletes the user with DN cn=2,ou=users,o=c_plusplus_project and displays the message: "User with DN 'cn=2,ou=users,o=c_plusplus_project' has been deleted successfully."
4. **Verifying Deletion in LDAP Directory**:
    - A screenshot of the LDAP directory structure shows that the user with ID 2 has been removed from the users organizational unit under the c_plusplus_project.
5. **Returning to LDAP User Management Menu**:
    - The application returns to the LDAP User Management Menu, ready for the next user action.

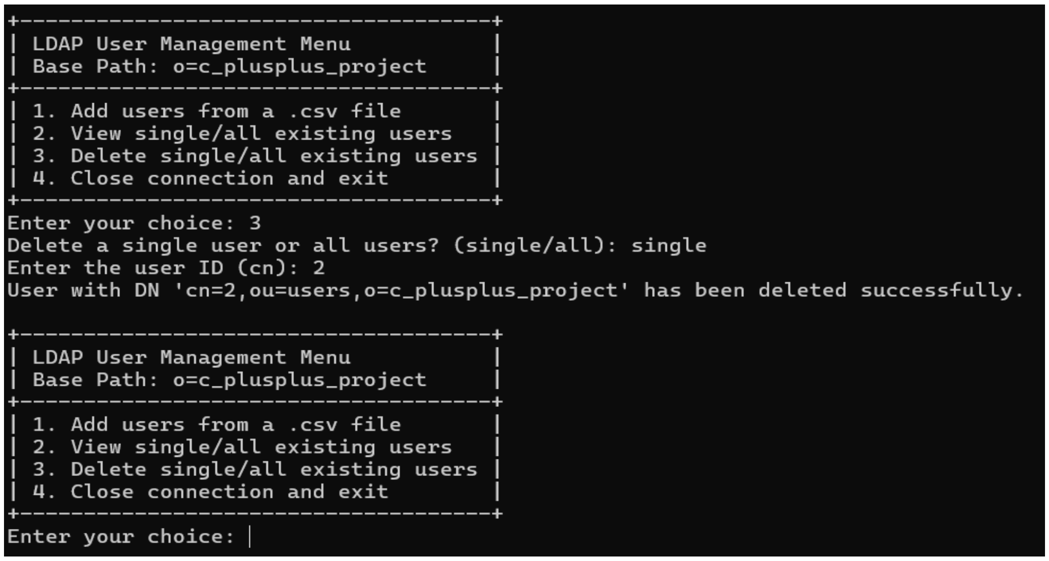
Why This Process:

- **User Deletion**: Demonstrates the application's ability to delete user entries from the LDAP server accurately.
- **Feedback and Confirmation**: Provides clear feedback to the user confirming the successful deletion of the specified user.
- **Verification**: Ensures that the changes are reflected in the LDAP directory, maintaining data integrity.

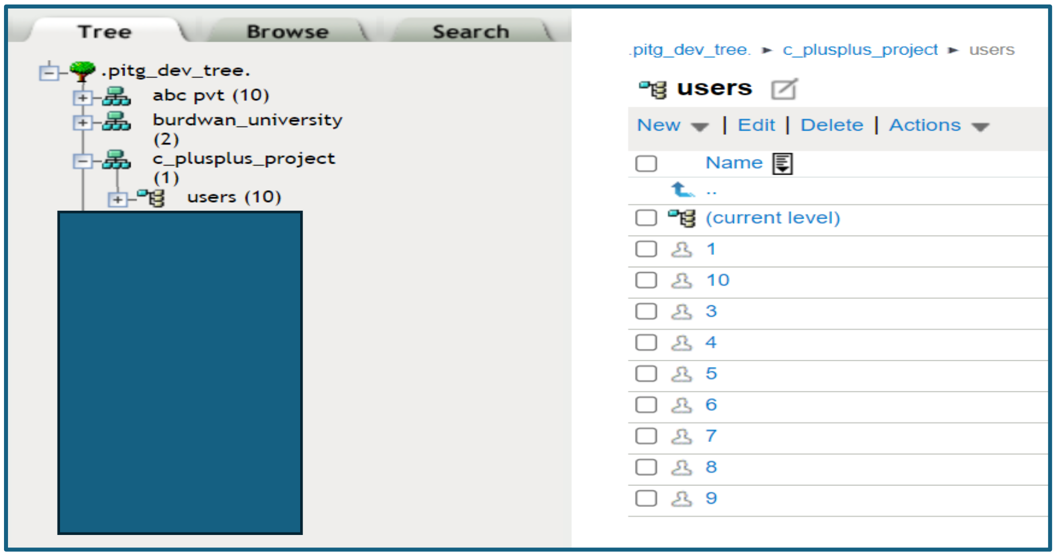
Additional Files:

1. **Directory Structure After Deletion**:
   - The directory structure shows the users organizational unit under the c_plusplus_project after the deletion of the user with ID 2.

This test case highlights the application's robustness in handling user deletions and ensuring that the changes are accurately reflected in the LDAP directory.



```
+------------------------------------+
| LDAP User Management Menu          |
| Base Path: o=c_plusplus_project    |
+------------------------------------+
| 1. Add users from a .csv file      |
| 2. View single/all existing users  |
| 3. Delete single/all existing users |
| 4. Close connection and exit       |
+------------------------------------+
Enter your choice: 3
Delete a single user or all users? (single/all): single
Enter the user ID (cn): 2
User with DN 'cn=2,ou=users,o=c_plusplus_project' has been deleted successfully.

+------------------------------------+
| LDAP User Management Menu          |
| Base Path: o=c_plusplus_project    |
+------------------------------------+
| 1. Add users from a .csv file      |
| 2. View single/all existing users  |
| 3. Delete single/all existing users |
| 4. Close connection and exit       |
+------------------------------------+
Enter your choice: |
```



.pitg_dev_tree.
abc pvt (10)
burdwan_university (2)
c_plusplus_project (1)
users (10)

[redacted]

.pitg_dev_tree. ► c_plusplus_project ► users

users

..
(current level)
1
10
3
4
5
6
7
8
9

7. Deleting All LDAP Users

Explanation:

In this test case, we demonstrate the process of deleting all LDAP users from the LDAP directory. This scenario tests the application's ability to remove multiple user entries accurately and ensure that the LDAP directory is empty after the operation.

Steps and Observations:

1. **Displaying LDAP User Management Menu**:
    - The application displays the LDAP User Management Menu with the following options:
        1. Add users from a .csv file
        2. View single/all existing users
        3. Delete single/all existing users
        4. Close connection and exit
2. **User Choice for Deleting Users**:
    - The user selects option 3 to delete users.
3. **All Users Deletion**:
    - The application prompts the user to specify whether to delete a single user or all users (single/all). The user chooses 'all'.
    - The application proceeds to delete all users, displaying messages for each user deletion:
        - "Deleting user with DN: cn=10,ou=users,o=c_plusplus_project"
        - "Deleting user with DN: cn=9,ou=users,o=c_plusplus_project"
        - "Deleting user with DN: cn=8,ou=users,o=c_plusplus_project"
        - "Deleting user with DN: cn=7,ou=users,o=c_plusplus_project"
        - "Deleting user with DN: cn=6,ou=users,o=c_plusplus_project"
        - "Deleting user with DN: cn=5,ou=users,o=c_plusplus_project"
        - "Deleting user with DN: cn=4,ou=users,o=c_plusplus_project"
        - "Deleting user with DN: cn=3,ou=users,o=c_plusplus_project"
        - "Deleting user with DN: cn=2,ou=users,o=c_plusplus_project"
        - "Deleting user with DN: cn=1,ou=users,o=c_plusplus_project"
    - The application confirms that all users have been deleted successfully with the message: "All users have been deleted successfully. Deleted all LDAP users under base path 'o=c_plusplus_project'."

4. **Verifying Deletion in LDAP Directory**:
    - A screenshot of the LDAP directory structure shows that the users organizational unit under the c_plusplus_project is empty, confirming the successful deletion of all users.
5. **Returning to LDAP User Management Menu**:
    - The application returns to the LDAP User Management Menu, ready for the next user action.
6. **Checking for Users Post-Deletion**:
    - The user selects option 2 to view users.
    - The application displays the message: "There are no users to view. Try adding users to the directory first."
    - The user selects option 3 to delete users.
    - The application displays the message: "There are no users to delete. Try adding users to the directory first."

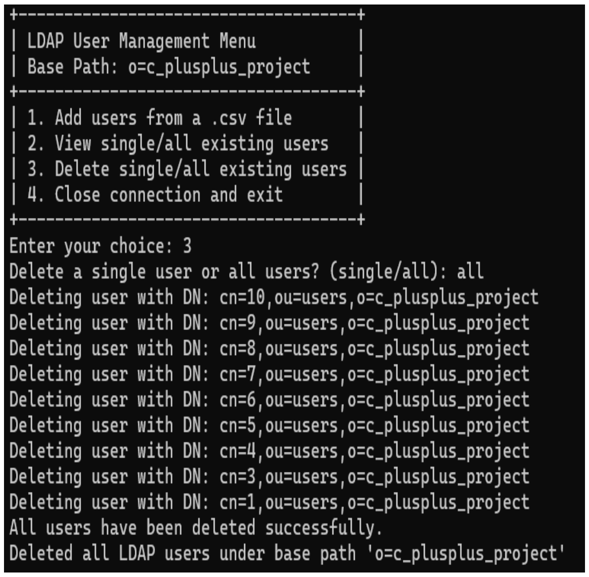
Why This Process:

- **Bulk Deletion**: Demonstrates the application's ability to delete multiple user entries from the LDAP server accurately.
- **Feedback and Confirmation**: Provides clear feedback to the user confirming the successful deletion of all users.
- **Verification**: Ensures that the changes are reflected in the LDAP directory, maintaining data integrity.
- **User Guidance**: Informs the user when no users are available for viewing or deletion, guiding them to add users first.

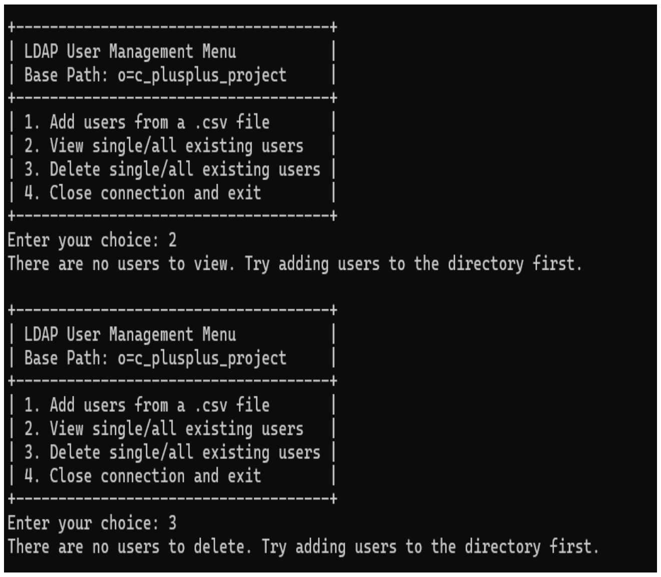
Additional Files:

1. **Directory Structure After Deletion**:
    - The directory structure shows the users organizational unit under the c_plusplus_project as empty after the deletion of all users.
2. **Messages Post-Deletion**:
    - Displays messages indicating that there are no users to view or delete, guiding the user to add users first.

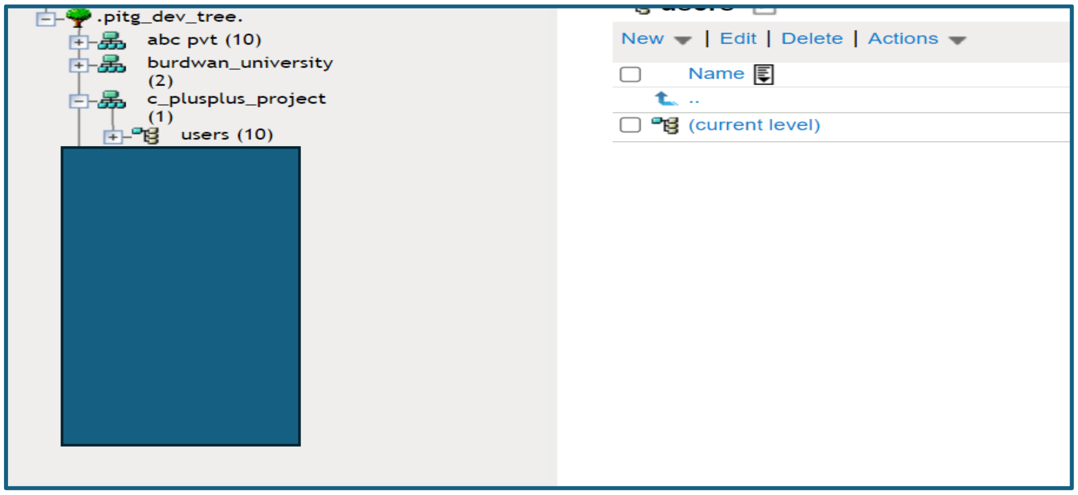
This test case highlights the application's robustness in handling bulk user deletions and ensuring that the changes are accurately reflected in the LDAP directory.



```
+--------------------------------------+
| LDAP User Management Menu            |
| Base Path: o=c_plusplus_project      |
+--------------------------------------+
| 1. Add users from a .csv file        |
| 2. View single/all existing users    |
| 3. Delete single/all existing users  |
| 4. Close connection and exit         |
+--------------------------------------+
Enter your choice: 3
Delete a single user or all users? (single/all): all
Deleting user with DN: cn=10,ou=users,o=c_plusplus_project
Deleting user with DN: cn=9,ou=users,o=c_plusplus_project
Deleting user with DN: cn=8,ou=users,o=c_plusplus_project
Deleting user with DN: cn=7,ou=users,o=c_plusplus_project
Deleting user with DN: cn=6,ou=users,o=c_plusplus_project
Deleting user with DN: cn=5,ou=users,o=c_plusplus_project
Deleting user with DN: cn=4,ou=users,o=c_plusplus_project
Deleting user with DN: cn=3,ou=users,o=c_plusplus_project
Deleting user with DN: cn=1,ou=users,o=c_plusplus_project
All users have been deleted successfully.
Deleted all LDAP users under base path 'o=c_plusplus_project'
```



```
+--------------------------------------+
| LDAP User Management Menu            |
| Base Path: o=c_plusplus_project      |
+--------------------------------------+
| 1. Add users from a .csv file        |
| 2. View single/all existing users    |
| 3. Delete single/all existing users  |
| 4. Close connection and exit         |
+--------------------------------------+
Enter your choice: 2
There are no users to view. Try adding users to the directory first.

+--------------------------------------+
| LDAP User Management Menu            |
| Base Path: o=c_plusplus_project      |
+--------------------------------------+
| 1. Add users from a .csv file        |
| 2. View single/all existing users    |
| 3. Delete single/all existing users  |
| 4. Close connection and exit         |
+--------------------------------------+
Enter your choice: 3
There are no users to delete. Try adding users to the directory first.
```



.pitg_dev_tree.
abc pvt (10)
burdwan_university (2)
c_plusplus_project (1)
users (10)
[redacted]

New | Edit | Delete | Actions

	Name
	..
	(current level)

8. Adding Users from Multiple CSV Files with Duplicate Data Handling

Explanation:

In this test case, we demonstrate the process of adding users from multiple CSV files to the LDAP directory, handling scenarios where duplicate user entries exist. This scenario tests the application's ability to detect and manage duplicate data entries and perform necessary operations to maintain data integrity.

Steps and Observations:

1. **Initial User Addition from CSV**:
   - The user adds users from the company.csv file, which contains 10 user entries.
   - The application successfully adds all users, confirming with the message: "All users successfully added: 1 2 3 4 5 6 7 8 9 10."
2. **Attempting to Add Users from Another CSV with Duplicate Entry**:
   - The user attempts to add users from the company2.csv file, which contains a single user entry identical to an existing user from company.csv.
   - The application detects the duplicate entry and displays an error message: "Error: CSV file does not contain any valid data rows. Returning to menu."
3. **Deleting the Duplicate User**:
   - The user chooses option 3 from the LDAP User Management Menu to delete a user.
   - The user specifies deleting a single user and provides the user ID: 1.
   - The application successfully deletes the user with DN cn=1,ou=users,o=c_plusplus_project, confirming with the message: "User with DN 'cn=1,ou=users,o=c_plusplus_project' has been deleted successfully."
4. **Adding Users from the CSV Again**:
   - The user attempts to add users from the company2.csv file again.
   - The application successfully adds the user, confirming with the message: "All users successfully added: 1."
5. **Verification**:
   - The LDAP directory now reflects the successful addition of users from both CSV files without any duplicates.

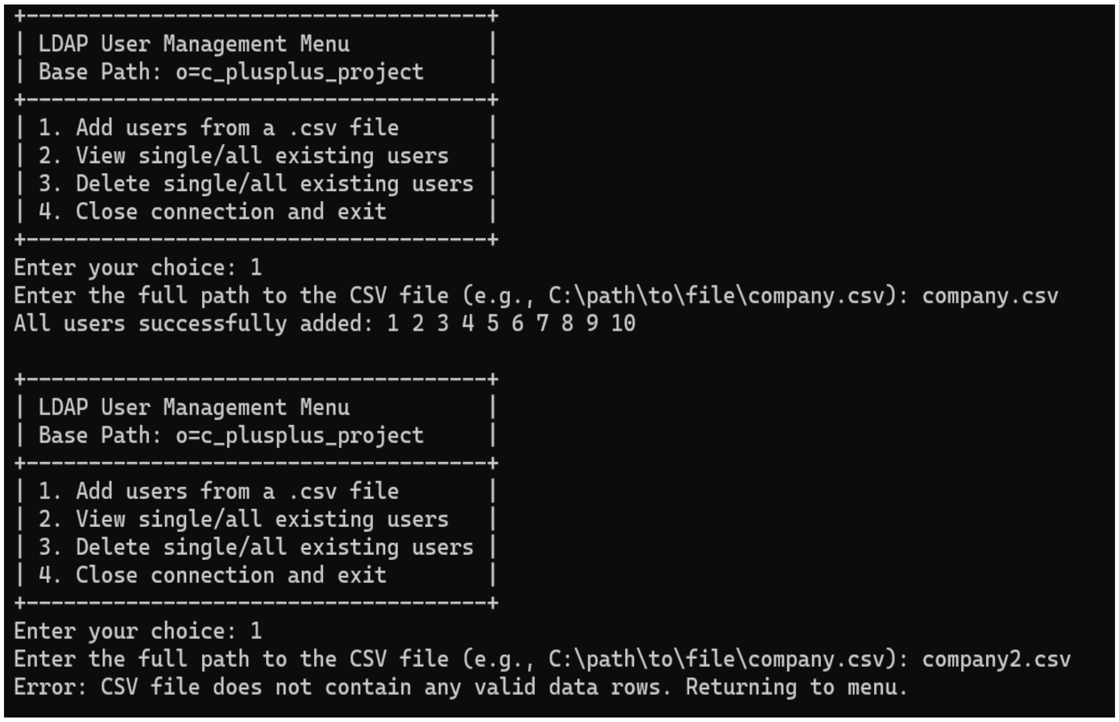
Why This Process:

- **Duplicate Handling**: Ensures that the application can detect and manage duplicate data entries, maintaining the integrity of the LDAP directory.
- **User Guidance**: Provides clear feedback and guidance to the user when duplicates are detected, enhancing user experience.
- **Data Integrity**: Demonstrates the application's ability to maintain data integrity by preventing duplicate entries and allowing for corrective actions.

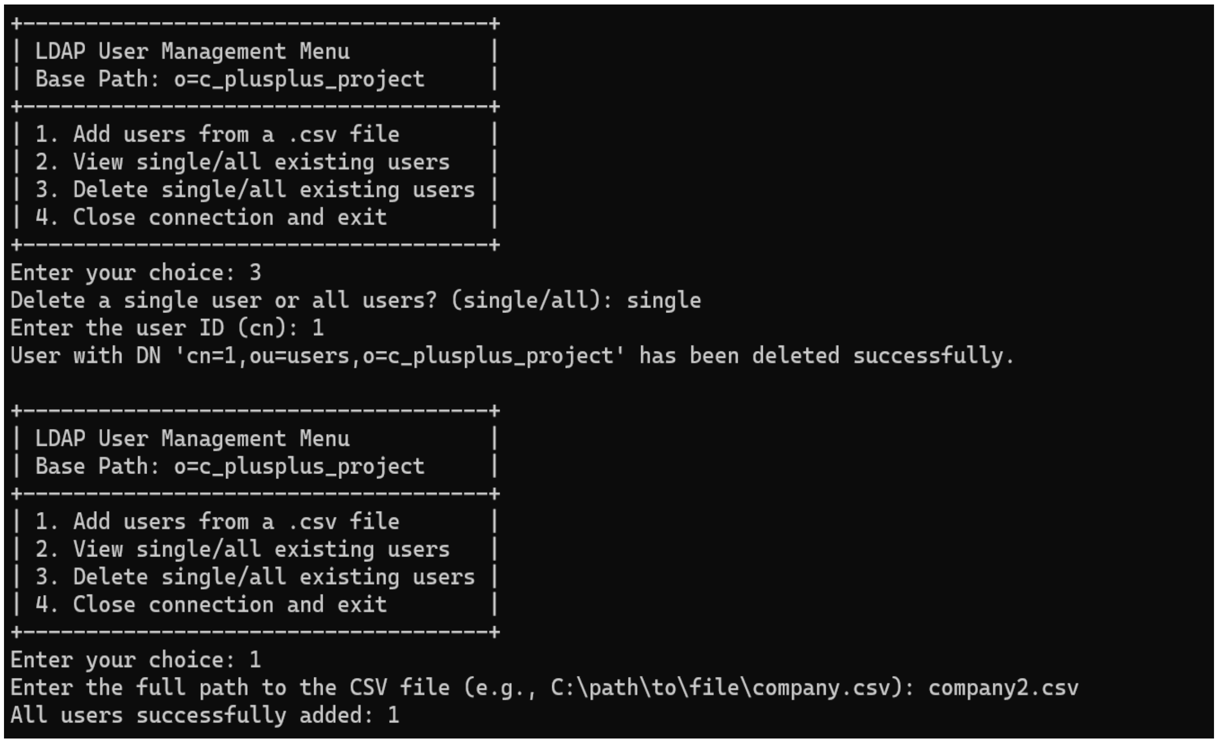
Additional Files:

1. **Screenshots of Initial User Addition**:
    - Displays the successful addition of users from the company.csv file.
2. **Screenshots of Duplicate Detection**:
    - Shows the error message when attempting to add duplicate user entries from the company2.csv file.
3. **Screenshots of User Deletion**:
    - Displays the successful deletion of the duplicate user entry.
4. **Screenshots of Successful Addition After Deletion**:
    - Shows the successful addition of the user from the company2.csv file after deleting the duplicate entry.

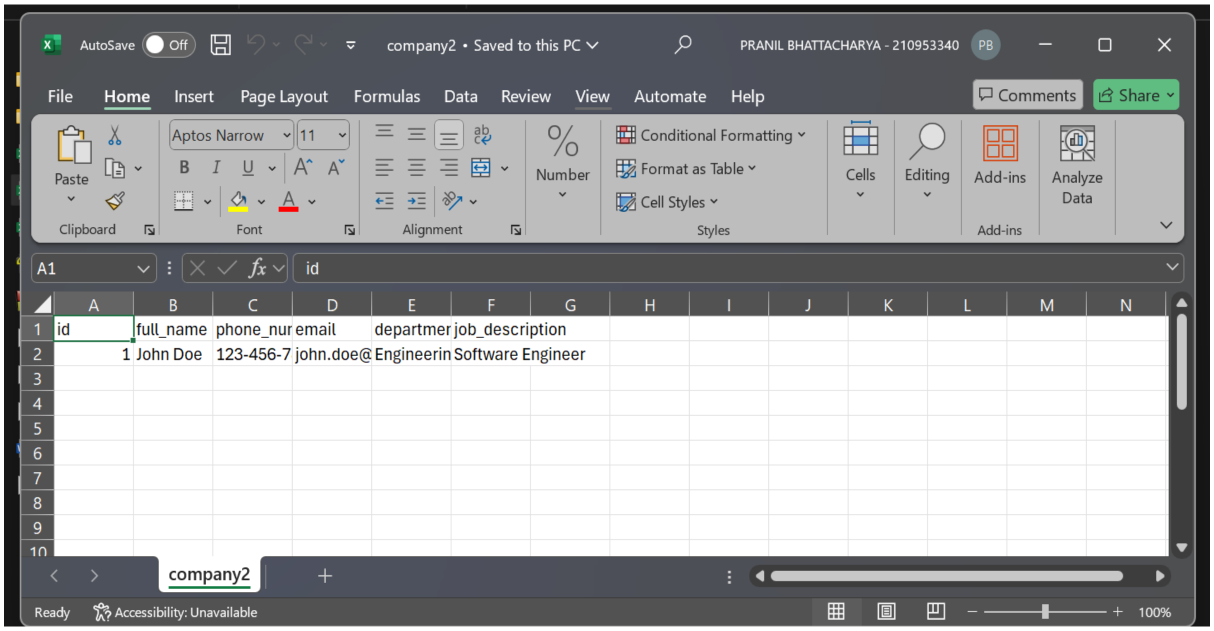
This test case highlights the application's robustness in handling duplicate user entries and maintaining the integrity of the LDAP directory.



```
+--------------------------------------+
| LDAP User Management Menu            |
| Base Path: o=c_plusplus_project      |
+--------------------------------------+
| 1. Add users from a .csv file        |
| 2. View single/all existing users    |
| 3. Delete single/all existing users  |
| 4. Close connection and exit         |
+--------------------------------------+
Enter your choice: 1
Enter the full path to the CSV file (e.g., C:\path\to\file\company.csv): company.csv
All users successfully added: 1 2 3 4 5 6 7 8 9 10

+--------------------------------------+
| LDAP User Management Menu            |
| Base Path: o=c_plusplus_project      |
+--------------------------------------+
| 1. Add users from a .csv file        |
| 2. View single/all existing users    |
| 3. Delete single/all existing users  |
| 4. Close connection and exit         |
+--------------------------------------+
Enter your choice: 1
Enter the full path to the CSV file (e.g., C:\path\to\file\company.csv): company2.csv
Error: CSV file does not contain any valid data rows. Returning to menu.
```



```
+------------------------------------+
| LDAP User Management Menu          |
| Base Path: o=c_plusplus_project    |
+------------------------------------+
| 1. Add users from a .csv file      |
| 2. View single/all existing users  |
| 3. Delete single/all existing users |
| 4. Close connection and exit       |
+------------------------------------+
Enter your choice: 3
Delete a single user or all users? (single/all): single
Enter the user ID (cn): 1
User with DN 'cn=1,ou=users,o=c_plusplus_project' has been deleted successfully.

+------------------------------------+
| LDAP User Management Menu          |
| Base Path: o=c_plusplus_project    |
+------------------------------------+
| 1. Add users from a .csv file      |
| 2. View single/all existing users  |
| 3. Delete single/all existing users |
| 4. Close connection and exit       |
+------------------------------------+
Enter your choice: 1
Enter the full path to the CSV file (e.g., C:\path\to\file\company.csv): company2.csv
All users successfully added: 1
```


AutoSave Off
company2 • Saved to this PC
PRANIL BHATTACHARYA - 210953340
File Home Insert Page Layout Formulas Data Review View Automate Help
Comments Share
Paste Clipboard Aptos Narrow 11 Font Alignment Number Conditional Formatting Format as Table Cell Styles Styles Cells Editing Add-ins Analyze Data
A1 id
company2
Ready Accessibility: Unavailable 100%

	A	B	C	D	E	F
1	id	full_name	phone_nur	email	departmer	job_description
2	1	John Doe	123-456-7	john.doe@	Engineerin	Software Engineer

9. Adding Users After Deleting Some Users from CSV File

Explanation:

In this test case, we demonstrate the process of adding users from a CSV file to the LDAP directory after some users have been deleted. This scenario tests the application's ability to detect and handle duplicate user entries, only adding users that do not already exist in the directory.

Steps and Observations:

1. **Initial User Addition from CSV**:
   - The user adds users from the company.csv file, which contains 10 user entries.
   - The application successfully adds all users, confirming with the message: "All users successfully added: 1 2 3 4 5 6 7 8 9 10."
2. **Deleting Some Users**:
   - The user deletes specific users using their user IDs.
   - User with ID 3 is deleted: "User with DN 'cn=3,ou=users,o=c_plusplus_project' has been deleted successfully."
   - User with ID 5 is deleted: "User with DN 'cn=5,ou=users,o=c_plusplus_project' has been deleted successfully."
   - User with ID 10 is deleted: "User with DN 'cn=10,ou=users,o=c_plusplus_project' has been deleted successfully."
3. **Adding Users from the Same CSV Again**:
   - The user attempts to add users from the company.csv file again.
   - The application detects the duplicate entries and displays messages for each duplicate user that could not be added, while successfully adding the previously deleted users:
     - "User ID: 1 – Reason: User already exists"
     - "User ID: 2 – Reason: User already exists"
     - "User ID: 4 – Reason: User already exists"
     - "User ID: 6 – Reason: User already exists"
     - "User ID: 7 – Reason: User already exists"
     - "User ID: 8 – Reason: User already exists"
     - "User ID: 9 – Reason: User already exists"
     - "Successfully added users: 3 5 10"

4. **Verification**:
    - The LDAP directory now reflects the successful addition of users 3, 5, and 10 without any duplicates from the company.csv file.

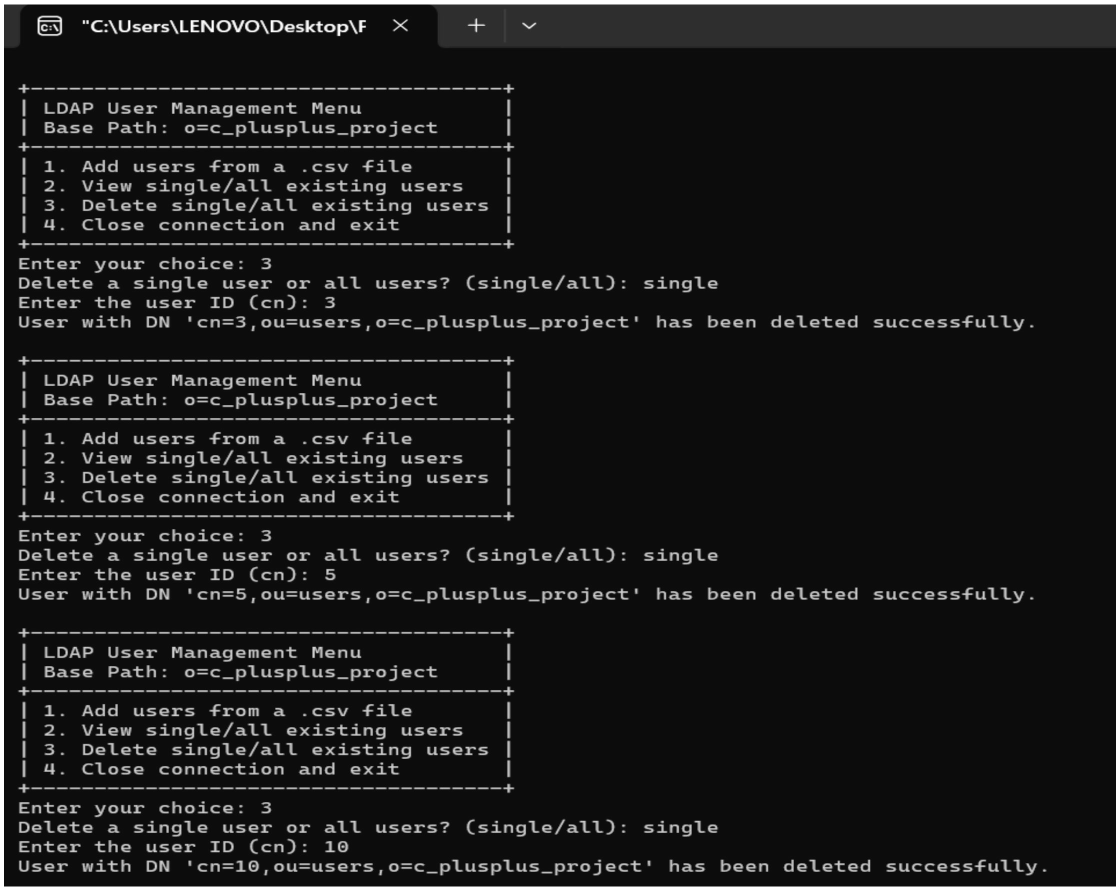
Why This Process:

- **Duplicate Handling**: Ensures that the application can detect and manage duplicate data entries, maintaining the integrity of the LDAP directory.
- **Selective Addition**: Demonstrates the application's ability to add only those users who do not already exist in the directory.
- **User Guidance**: Provides clear feedback and guidance to the user regarding which users were added and which were not due to duplication.

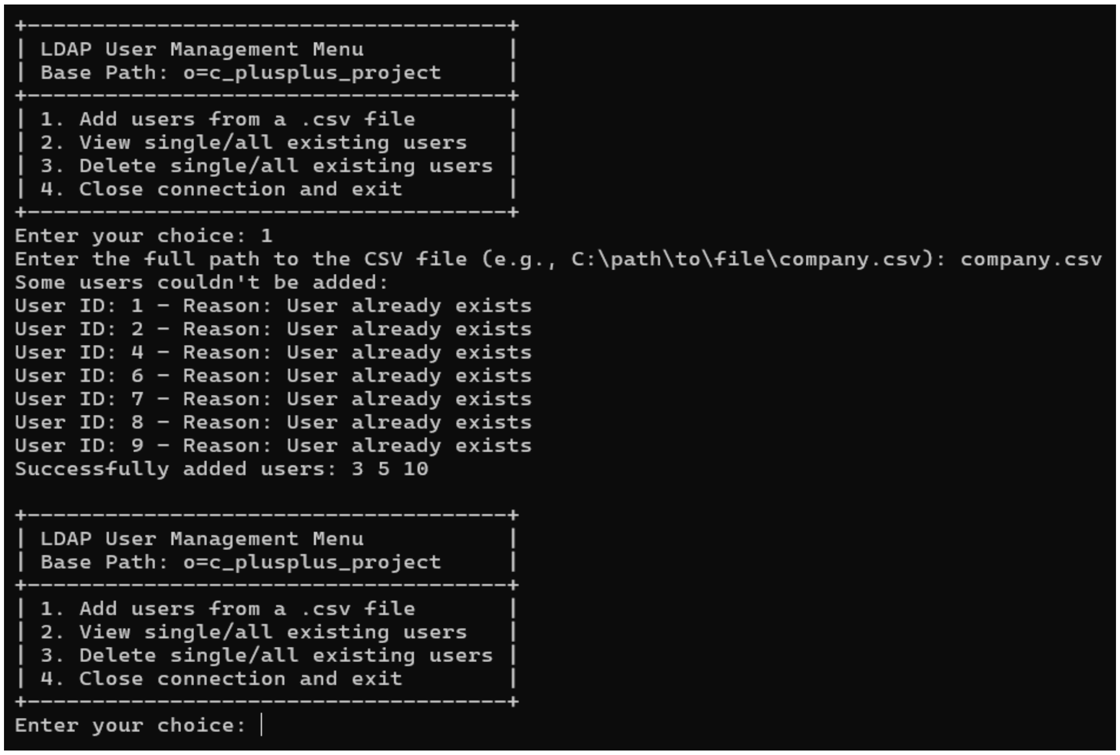
Additional Files:

1. **Screenshots of Initial User Addition**:
    - Displays the successful addition of users from the company.csv file.
2. **Screenshots of User Deletion**:
    - Shows the successful deletion of specific user entries.
3. **Screenshots of Adding Users After Deletion**:
    - Displays the messages indicating successful addition of deleted users and detection of duplicate entries.

This test case highlights the application's robustness in handling user entries from CSV files, ensuring data integrity by preventing duplicates and selectively adding users based on their existence in the LDAP directory.



```
"C:\Users\LENOVO\Desktop\F

+------------------------------------+
| LDAP User Management Menu          |
| Base Path: o=c_plusplus_project    |
+------------------------------------+
| 1. Add users from a .csv file      |
| 2. View single/all existing users  |
| 3. Delete single/all existing users|
| 4. Close connection and exit       |
+------------------------------------+
Enter your choice: 3
Delete a single user or all users? (single/all): single
Enter the user ID (cn): 3
User with DN 'cn=3,ou=users,o=c_plusplus_project' has been deleted successfully.

+------------------------------------+
| LDAP User Management Menu          |
| Base Path: o=c_plusplus_project    |
+------------------------------------+
| 1. Add users from a .csv file      |
| 2. View single/all existing users  |
| 3. Delete single/all existing users|
| 4. Close connection and exit       |
+------------------------------------+
Enter your choice: 3
Delete a single user or all users? (single/all): single
Enter the user ID (cn): 5
User with DN 'cn=5,ou=users,o=c_plusplus_project' has been deleted successfully.

+------------------------------------+
| LDAP User Management Menu          |
| Base Path: o=c_plusplus_project    |
+------------------------------------+
| 1. Add users from a .csv file      |
| 2. View single/all existing users  |
| 3. Delete single/all existing users|
| 4. Close connection and exit       |
+------------------------------------+
Enter your choice: 3
Delete a single user or all users? (single/all): single
Enter the user ID (cn): 10
User with DN 'cn=10,ou=users,o=c_plusplus_project' has been deleted successfully.
```



```
+------------------------------------+
| LDAP User Management Menu          |
| Base Path: o=c_plusplus_project    |
+------------------------------------+
| 1. Add users from a .csv file      |
| 2. View single/all existing users  |
| 3. Delete single/all existing users|
| 4. Close connection and exit       |
+------------------------------------+
Enter your choice: 1
Enter the full path to the CSV file (e.g., C:\path\to\file\company.csv): company.csv
Some users couldn't be added:
User ID: 1 - Reason: User already exists
User ID: 2 - Reason: User already exists
User ID: 4 - Reason: User already exists
User ID: 6 - Reason: User already exists
User ID: 7 - Reason: User already exists
User ID: 8 - Reason: User already exists
User ID: 9 - Reason: User already exists
Successfully added users: 3 5 10

+------------------------------------+
| LDAP User Management Menu          |
| Base Path: o=c_plusplus_project    |
+------------------------------------+
| 1. Add users from a .csv file      |
| 2. View single/all existing users  |
| 3. Delete single/all existing users|
| 4. Close connection and exit       |
+------------------------------------+
Enter your choice:
```

10. Exiting the LDAP User Management Application

Explanation:

In this test case, we demonstrate the process of exiting the LDAP User Management Application. This scenario tests the application's ability to properly unbind from the LDAP server and close the connection, ensuring a clean exit from the program.

Steps and Observations:

1. **Displaying LDAP User Management Menu**:
   - The application displays the LDAP User Management Menu with the following options:
     1. Add users from a .csv file
     2. View single/all existing users
     3. Delete single/all existing users
     4. Close connection and exit
2. **User Choice to Exit**:
   - The user selects option 4 to close the connection and exit the application.
3. **Unbinding from LDAP Server**:
   - The application initiates the unbinding process from the LDAP server, displaying the message: "Unbinding from LDAP server...".
   - The unbinding is successful, and the connection is closed, confirmed by the message: "LDAP unbind successful. Connection closed."
4. **Confirming Exit**:
   - The application prompts the user to confirm if they want to reconnect to the LDAP server again with the question: "Do you want to connect to the LDAP server again? (y/n)". The user chooses 'n'.
   - The application then prompts the user to confirm if they are sure they want to exit with the question: "Are you sure you want to exit? (y/n)". The user chooses 'y'.
5. **Exiting the Program**:
   - The application exits, displaying the message: "Exiting the program. Goodbye!".

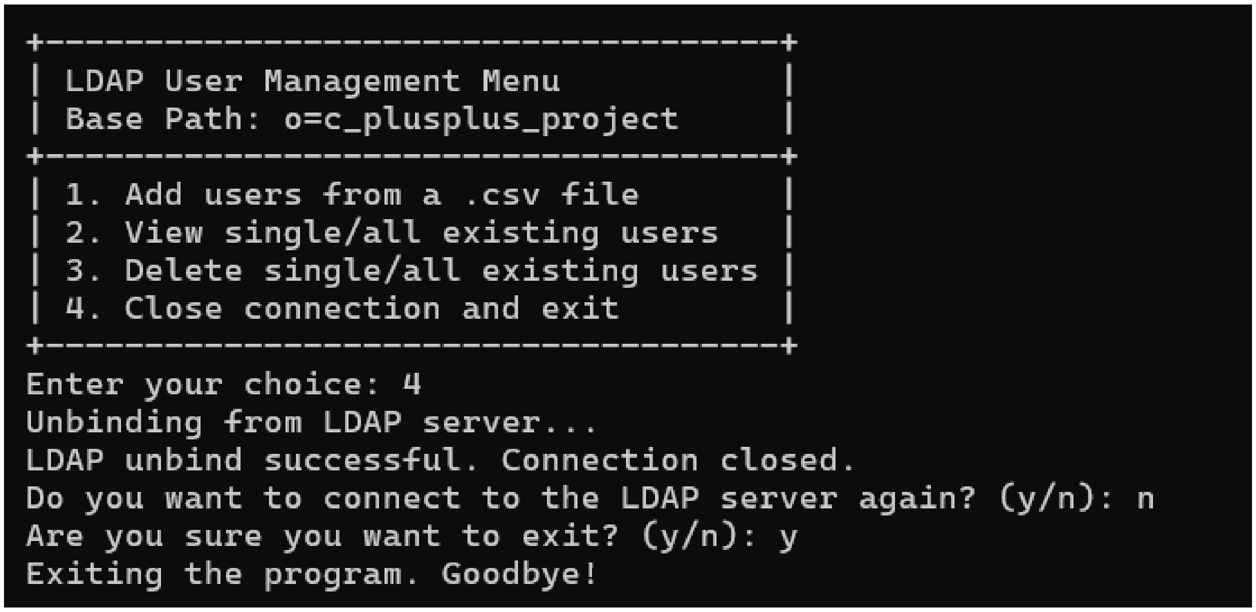
Why This Process:

- **Proper Unbinding**: Ensures that the application properly unbinds from the LDAP server and closes the connection to avoid any potential issues or resource leaks.
- **User Confirmation**: Provides clear prompts for the user to confirm their intention to exit, enhancing user experience and preventing accidental exits.
- **Clean Exit**: Demonstrates the application's ability to exit cleanly and provide a proper goodbye message to the user.

Additional File:

1. **Screenshot of Exit Process**:
    - Displays the successful unbinding from the LDAP server and the clean exit process from the application.

This test case highlights the application's robustness in handling the exit process, ensuring a proper and clean disconnection from the LDAP server and closing the program with user confirmation.



```
+--------------------------------------+
| LDAP User Management Menu            |
| Base Path: o=c_plusplus_project      |
+--------------------------------------+
| 1. Add users from a .csv file        |
| 2. View single/all existing users    |
| 3. Delete single/all existing users |
| 4. Close connection and exit         |
+--------------------------------------+
Enter your choice: 4
Unbinding from LDAP server...
LDAP unbind successful. Connection closed.
Do you want to connect to the LDAP server again? (y/n): n
Are you sure you want to exit? (y/n): y
Exiting the program. Goodbye!
```

VIII. Appendix

Source Code with Simple Comments:

```
#include <iostream>     // For input and output operations
#include <fstream>      // For file handling
#include <sstream>      // For string stream operations
#include <Windows.h>    // For Windows-specific functions
#include <Winldap.h>    // For LDAP functions
#include <string>       // For string operations
#include <vector>       // For storing user data
#include <map>          // For clustering errors
#include <algorithm>    // For sorting

using namespace std;

// Link the Wldap32 library for LDAP functions
#pragma comment(lib, "Wldap32.lib")

// Function to print sensitive information safely
void printSensitiveInfo(const string& info)
{
    cout << "Binding with DN: " << info << endl;
}

// Function to add a single LDAP user
int addLDAPUser(LDAP* ldap, const string& id, const string& fullName, const
string& phoneNumber, const string& email, const string& department, const
string& jobDescription)
{
    int rc = LDAP_SUCCESS;

    // Split the full name into first name and last name
    istringstream iss(fullName);
    string firstName, lastName;
    iss >> firstName;
    getline(iss, lastName);

    // Construct the Distinguished Name (DN) for the new user
    string newUserDN = "cn=" + id + ",ou=users,o=c_plusplus_project";

    // Prepare the attributes for the new user
      LDAPMod  mod_cn,  mod_sn,  mod_givenName,  mod_mail,  mod_objectClass,
mod_department, mod_phoneNumber, mod_jobDescription;
    LDAPMod* mods[9];

    // Set values for each attribute
    char* cn_values[] = { const_cast<char*>(id.c_str()), nullptr };
    char* sn_values[] = { const_cast<char*>(lastName.c_str()), nullptr };
      char*  givenName_values[]  =  {  const_cast<char*>(firstName.c_str()),
nullptr };
    char* mail_values[] = { const_cast<char*>(email.c_str()), nullptr };
      char*  objectClass_values[]  =  {  const_cast<char*>("inetOrgPerson"),
const_cast<char*>("organizationalPerson"),      const_cast<char*>("person"),
const_cast<char*>("top"), nullptr };
```

```
      char*  department_values[]  =  {  const_cast<char*>(department.c_str()),
nullptr };
     char*  phoneNumber_values[]  =  {  const_cast<char*>(phoneNumber.c_str()),
nullptr };
                    char*          jobDescription_values[]          =          {
const_cast<char*>(jobDescription.c_str()), nullptr };

    // Fill in LDAPMod structures
    mod_cn.mod_op = LDAP_MOD_ADD;
    mod_cn.mod_type = const_cast<char*>("cn");
    mod_cn.mod_values = cn_values;

    mod_sn.mod_op = LDAP_MOD_ADD;
    mod_sn.mod_type = const_cast<char*>("sn");
    mod_sn.mod_values = sn_values;

    mod_givenName.mod_op = LDAP_MOD_ADD;
    mod_givenName.mod_type = const_cast<char*>("givenName");
    mod_givenName.mod_values = givenName_values;

    mod_mail.mod_op = LDAP_MOD_ADD;
    mod_mail.mod_type = const_cast<char*>("mail");
    mod_mail.mod_values = mail_values;

    mod_objectClass.mod_op = LDAP_MOD_ADD;
    mod_objectClass.mod_type = const_cast<char*>("objectClass");
    mod_objectClass.mod_values = objectClass_values;

    mod_department.mod_op = LDAP_MOD_ADD;
    mod_department.mod_type = const_cast<char*>("ou");
    mod_department.mod_values = department_values;

    mod_phoneNumber.mod_op = LDAP_MOD_ADD;
    mod_phoneNumber.mod_type = const_cast<char*>("telephoneNumber");
    mod_phoneNumber.mod_values = phoneNumber_values;

    mod_jobDescription.mod_op = LDAP_MOD_ADD;
    mod_jobDescription.mod_type = const_cast<char*>("description");
    mod_jobDescription.mod_values = jobDescription_values;

    // Add all attributes to the mods array
    mods[0] = &mod_cn;
    mods[1] = &mod_sn;
    mods[2] = &mod_givenName;
    mods[3] = &mod_mail;
    mods[4] = &mod_objectClass;
    mods[5] = &mod_department;
    mods[6] = &mod_phoneNumber;
    mods[7] = &mod_jobDescription;
    mods[8] = nullptr;

    // Perform the add operation
     rc  =  ldap_add_ext_sA(ldap,  const_cast<char*>(newUserDN.c_str()),  mods,
nullptr, nullptr);
    return rc;
}
```

```
// Function to check if an LDAP user exists
bool userExists(LDAP* ldap, const string& userDN)
{
    int rc = LDAP_SUCCESS;
    LDAPMessage* result = nullptr;

    // Search for the user in the LDAP directory
        rc = ldap_search_ext_sA(ldap, const_cast<char*>(userDN.c_str()),
LDAP_SCOPE_BASE, const_cast<char*>("(objectClass=inetOrgPerson)"), nullptr,
0, nullptr, nullptr, nullptr, LDAP_NO_LIMIT, &result);

    if (rc == LDAP_SUCCESS && ldap_count_entries(ldap, result) > 0)
    {
        ldap_msgfree(result);
        return true;
    }

    ldap_msgfree(result);
    return false;
}

// Function to delete all LDAP users under a specific path
int deleteAllLDAPUsers(LDAP* ldap, const string& basePath)
{
    int rc = LDAP_SUCCESS;

    LDAPMessage* result = nullptr;
    LDAPMessage* entry = nullptr;
    string filter = "(objectClass=inetOrgPerson)";
    char* attrs[] = { const_cast<char*>("cn"), nullptr };

    // Construct the search base
    string searchBase = "ou=users," + basePath;

    // Search for all users under the specified base path
      rc = ldap_search_ext_sA(ldap, const_cast<char*>(searchBase.c_str()),
LDAP_SCOPE_ONELEVEL, const_cast<char*>(filter.c_str()), attrs, 0, nullptr,
nullptr, nullptr, LDAP_NO_LIMIT, &result);

    if (rc != LDAP_SUCCESS)
    {
        cerr << "LDAP search failed: " << ldap_err2stringA(rc) << endl;
        return rc;
    }

    // Check if the user list is empty
    if (ldap_count_entries(ldap, result) == 0)
    {
          cout << "There are no users to delete. Try adding users to the
directory first." << endl;
        ldap_msgfree(result);
        return rc;
    }

    // Iterate through the search results and delete each user
     for (entry = ldap_first_entry(ldap, result); entry != nullptr; entry =
ldap_next_entry(ldap, entry))
```

```
    {
        char* dn = ldap_get_dnA(ldap, entry);
        cout << "Deleting user with DN: " << dn << endl;

        rc = ldap_delete_ext_sA(ldap, dn, nullptr, nullptr);
        if (rc != LDAP_SUCCESS)
        {
             cerr << "Failed to delete user with DN '" << dn << "': " <<
ldap_err2stringA(rc) << endl;
            ldap_memfreeA(dn);
            continue;
        }

        ldap_memfreeA(dn);
    }

    ldap_msgfree(result);
    cout << "All users have been deleted successfully." << endl;

    return rc;
}

// Function to delete a single LDAP user by user ID
int deleteSingleLDAPUser(LDAP* ldap, const string& userDN)
{
    int rc = LDAP_SUCCESS;

    // Check if the user exists before attempting to delete
    if (!userExists(ldap, userDN))
    {
        cout << "User with DN '" << userDN << "' does not exist." << endl;
        return LDAP_NO_SUCH_OBJECT;
    }

    rc = ldap_delete_ext_sA(ldap, const_cast<char*>(userDN.c_str()), nullptr,
nullptr);
    if (rc != LDAP_SUCCESS)
    {
         cerr << "Failed to delete user with DN '" << userDN << "': " <<
ldap_err2stringA(rc) << endl;
    }
    else
    {
       cout << "User with DN '" << userDN << "' has been deleted successfully."
<< endl;
    }

    return rc;
}

// Function to check if a line is properly comma-delimited and matches the
expected format
bool isProperlyFormatted(const string& line)
{
    istringstream iss(line);
    string token;
    int columnCount = 0;
    while (getline(iss, token, ','))
    {
        columnCount++;
    }

    return columnCount == 6;
}

// Function to display detailed information of a single LDAP user
void displaySingleLDAPUser(LDAP* ldap, const string& userDN)
{
    int rc = LDAP_SUCCESS;

    LDAPMessage* result = nullptr;
    LDAPMessage* entry = nullptr;
     char*  attrs[]  =  {  const_cast<char*>("cn"),  const_cast<char*>("sn"),
const_cast<char*>("givenName"),                      const_cast<char*>("mail"),
const_cast<char*>("ou"),                const_cast<char*>("telephoneNumber"),
const_cast<char*>("description"), nullptr };

    // Search for the user in the LDAP directory
       rc   =   ldap_search_ext_sA(ldap,   const_cast<char*>(userDN.c_str()),
LDAP_SCOPE_BASE, const_cast<char*>("(objectClass=inetOrgPerson)"), attrs, 0,
nullptr, nullptr, nullptr, LDAP_NO_LIMIT, &result);

    if (rc != LDAP_SUCCESS)
    {
        cerr << "LDAP search failed: " << ldap_err2stringA(rc) << endl;
        return;
    }

    entry = ldap_first_entry(ldap, result);
    if (entry != nullptr)
    {
        cout << "\nUser Details (DN: " << userDN << "):\n";
        for (int i = 0; attrs[i] != nullptr; i++)
        {
            char* attr = attrs[i];
            BerElement* ber = nullptr;
            char** values = ldap_get_valuesA(ldap, entry, attr);
            if (values)
            {
                cout << attr << ": " << values[0] << endl;
                ldap_value_freeA(values);
            }
        }
    }
    else
    {
        cout << "No user found with DN: " << userDN << endl;
    }

    ldap_msgfree(result);
}

// Function to display all LDAP users under a specific path
```

```
void displayAllLDAPUsers(LDAP* ldap, const string& basePath)
{
    int rc = LDAP_SUCCESS;

    LDAPMessage* result = nullptr;
    LDAPMessage* entry = nullptr;
    string filter = "(objectClass=inetOrgPerson)";
     char*  attrs[]  =  {  const_cast<char*>("cn"),  const_cast<char*>("sn"),
const_cast<char*>("givenName"),                    const_cast<char*>("mail"),
const_cast<char*>("ou"),                 const_cast<char*>("telephoneNumber"),
const_cast<char*>("description"), nullptr };

    // Construct the search base
    string searchBase = "ou=users," + basePath;

    // Search for all users under the specified base path
      rc  =  ldap_search_ext_sA(ldap,  const_cast<char*>(searchBase.c_str()),
LDAP_SCOPE_ONELEVEL,  const_cast<char*>(filter.c_str()),  attrs,  0,  nullptr,
nullptr, nullptr, LDAP_NO_LIMIT, &result);

    if (rc != LDAP_SUCCESS)
    {
        cerr << "LDAP search failed: " << ldap_err2stringA(rc) << endl;
        return;
    }

    // Check if the user list is empty
    if (ldap_count_entries(ldap, result) == 0)
    {
         cout  <<  "There  are  no  users  to  display.  Try  adding  users  to  the
directory first." << endl;
        ldap_msgfree(result);
        return;
    }

    // Store users in a vector to sort them by ID
    vector<string> users;
     for  (entry = ldap_first_entry(ldap, result); entry != nullptr; entry =
ldap_next_entry(ldap, entry))
    {
        char* dn = ldap_get_dnA(ldap, entry);
        users.push_back(dn);
        ldap_memfreeA(dn);
    }

    // Sort users by their IDs
    sort(users.begin(), users.end());

    // Display sorted users
    cout << "\nExisting LDAP users under " << searchBase << ":\n";
    for (const auto& user : users)
    {
        cout << "\nUser Details (DN: " << user << "):\n";
           rc  =  ldap_search_ext_sA(ldap,  const_cast<char*>(user.c_str()),
LDAP_SCOPE_BASE, const_cast<char*>("(objectClass=inetOrgPerson)"), attrs, 0,
nullptr, nullptr, nullptr, LDAP_NO_LIMIT, &result);
        if (rc == LDAP_SUCCESS)
        {
            entry = ldap_first_entry(ldap, result);
            for (int i = 0; attrs[i] != nullptr; i++)
            {
                char* attr = attrs[i];
                BerElement* ber = nullptr;
                char** values = ldap_get_valuesA(ldap, entry, attr);
                if (values)
                {
                    cout << attr << ": " << values[0] << endl;
                    ldap_value_freeA(values);
                }
            }
        }
        ldap_msgfree(result);
    }
}

int main()
{
    // Display application purpose
    cout << "Welcome to the LDAP User Management Application.\n";
    cout << "This application allows you to manage LDAP users, including
adding, viewing, and deleting users.\n";
    cout << "Please follow the prompts to perform the desired operations.\n"
<< endl;

    LDAP* ldap = nullptr;
    int rc = 0;
    string connectChoice;
    bool firstAttempt = true;

    // LDAP server details
    const char* ldapHost = "xxx.xxx.x.x"; // hidden here for security purposes
    int ldapPort = 389;
    const char* ldapUsername = "cn=idamadmin,ou=sa,o=pitg";
    const char* ldapPassword = "xxxxxxxxxxx"; // hidden here for security
purposes
    string basePath = "o=c_plusplus_project";

    while (true)
    {
        // Prompt user to connect to LDAP server
        if (firstAttempt)
        {
            cout << "Do you want to connect to the LDAP server? (y/n): ";
        }
        else
        {
            cout << "Do you want to connect to the LDAP server again? (y/n):
";
        }
        getline(cin, connectChoice);

        // Convert input to lowercase for case-insensitive comparison
                    transform(connectChoice.begin(),    connectChoice.end(),
connectChoice.begin(), ::tolower);
```

```
        if (connectChoice == "y" || connectChoice == "yes")
        {
            firstAttempt = false;
            cout << "Attempting to initialize LDAP connection..." << endl;

            // Initialize LDAP connection
            ldap = ldap_initA(const_cast<char*>(ldapHost), ldapPort);
            if (ldap == nullptr)
            {
                 cerr << "Failed to initialize LDAP connection. Please try
again later." << endl;
                continue;
            }
            cout << "LDAP connection initialized successfully." << endl;

            cout << "Setting LDAP protocol version..." << endl;

            // Set LDAP options
            ULONG version = LDAP_VERSION3;
             cout << "Setting LDAP option: LDAP_OPT_PROTOCOL_VERSION = " <<
version << endl;
                   rc  =  ldap_set_option(ldap,  LDAP_OPT_PROTOCOL_VERSION,
reinterpret_cast<void*>(&version));
            if (rc != LDAP_SUCCESS)
            {
                    cerr  <<  "Failed  to  set  LDAP  protocol  version:  "  <<
ldap_err2stringA(rc) << endl;
                ldap_unbind_s(ldap);
                continue;
            }
            cout << "LDAP protocol version set successfully." << endl;

            cout << "Attempting LDAP bind..." << endl;

            // Bind to LDAP server (authenticate)
            printSensitiveInfo(ldapUsername);
             rc = ldap_simple_bind_sA(ldap, const_cast<char*>(ldapUsername),
const_cast<char*>(ldapPassword));
            if (rc != LDAP_SUCCESS)
            {
               cerr << "LDAP bind failed: " << ldap_err2stringA(rc) << endl;
                ldap_unbind_s(ldap);
                cout << "Please try again later." << endl;
                continue;
            }
            cout << "LDAP bind successful." << endl;

            // Menu-driven interface
            string choice;
            while (true)
            {
                // Display the main menu
                cout << "\n+-------------------------------------+\n";
                cout << "| LDAP User Management Menu           |\n";
                cout << "| Base Path: " << basePath << "      |\n";
                cout << "+-------------------------------------+\n";
                cout << "| 1. Add users from a .csv file       |\n";
                cout << "| 2. View single/all existing users   |\n";
                cout << "| 3. Delete single/all existing users |\n";
                cout << "| 4. Close connection and exit        |\n";
                cout << "+-------------------------------------+\n";
                cout << "Enter your choice: ";
                getline(cin, choice);

                if (choice == "1")
                {
                    // Add users from a .csv file
                    string filePath;

                    while (true)
                    {
                        // Prompt user for the path to the CSV file
                         cout << "Enter the full path to the CSV file (e.g.,
C:\\path\\to\\file\\company.csv): ";
                        getline(cin, filePath);

                        // Check if the file path is valid
                        if (filePath.empty())
                        {
                             cerr << "Error: The file path is empty. Please
enter the correct file again." << endl;
                            continue;
                        }

                        // Check if the file exists
                        ifstream file(filePath);
                        if (!file)
                        {
                            cerr << "Error: The file does not exist. Please
enter the correct file again." << endl;
                            continue;
                        }

                        // Check if the file is a CSV file
                         if (filePath.substr(filePath.find_last_of(".") + 1)
!= "csv")
                        {
                          cerr << "Error: The file is not a CSV file. Please
enter the correct file again." << endl;
                            continue;
                        }

                        string line;
                        bool headerChecked = false;
                        bool properFormat = true;
                        bool hasValidDataRow = false;

                        // Structure to store user results
                        struct UserResult
                        {
                            string id;
                            string error;
                        };
                        vector<UserResult> results;
                        vector<string> addedUsers;

                        // Read the CSV file line by line
                        while (getline(file, line))
                        {
                            if (!headerChecked)
                            {
                                // Check if the header is correct
                                                              if   (line   !=
"id,full_name,phone_number,email,department,job_description")
                                {
                                        cerr << "Error: CSV file header is
incorrect. Returning to menu." << endl;
                                    properFormat = false;
                                    break;
                                }
                                headerChecked = true;
                                continue;
                            }

                            // Check if the line is properly formatted
                            if (!isProperlyFormatted(line))
                            {
                                 cerr << "Error: File is not properly comma-
delimited. Returning to menu." << endl;
                                properFormat = false;
                                break;
                            }

                            // Extract user details from the line
                         string id, fullName, phoneNumber, email, department,
jobDescription;
                            istringstream iss(line);
                           if (getline(getline(iss, id, ','), fullName, ',')
&&
                                        getline(getline(getline(getline(iss,
phoneNumber, ','), email, ','), department, ','), jobDescription, ','))
                            {
                                 string userDN = "cn=" + id + ",ou=users," +
basePath;
                                if (userExists(ldap, userDN))
                                {
                                       results.push_back({ id, "User already
exists" });
                                }
                                else
                                {
                                       rc = addLDAPUser(ldap, id, fullName,
phoneNumber, email, department, jobDescription);
                                    if (rc != LDAP_SUCCESS)
                                    {
                                                    results.push_back({  id,
ldap_err2stringA(rc) });
                                    }
                                    else
                                    {
                                        hasValidDataRow = true;
                                        addedUsers.push_back(id);
                                    }
                                }
                            }
                        }
                        file.close();

                        // Display results of adding users
                        if (properFormat && !hasValidDataRow)
                        {
                              cerr << "Error: CSV file does not contain any
valid data rows. Returning to menu." << endl;
                        }
                        else if (results.empty())
                        {
                            cout << "All users successfully added: ";
                            for (const auto& user : addedUsers)
                            {
                                cout << user << " ";
                            }
                            cout << endl;
                        }
                        else if (addedUsers.empty())
                        {
                            bool sameError = true;
                            string commonError = results[0].error;
                            for (const auto& result : results)
                            {
                                if (result.error != commonError)
                                {
                                    sameError = false;
                                    break;
                                }
                            }
                            if (sameError)
                            {
                                cout << "All users can't be added due to the
same reason: " << commonError << endl;
                            }
                            else
                            {
                                map<string, vector<string>> clusteredErrors;
                                for (const auto& result : results)
                                {
                                    clusteredErrors[result.error].push_back(
result.id);
                                }
                                cout << "All users can't be added due to the
following reasons:" << endl;
                                for (const auto& error : clusteredErrors)
                                {
                                    cout << "Reason: " << error.first << " -
Users: ";
                                    for (const auto& id : error.second)
                                    {
                                        cout << id << " ";
                                    }
                                    cout << endl;
                                }
                            }
                        }
                        else
                        {
                            cout << "Some users couldn't be added:" << endl;
                            for (const auto& result : results)
                            {
                                   cout << "User ID: " << result.id << " - 
Reason: " << result.error << endl;
                            }
                            cout << "Successfully added users: ";
                            for (const auto& user : addedUsers)
                            {
                                cout << user << " ";
                            }
                            cout << endl;
                        }

                        break;
                    }
                }
                else if (choice == "2")
                {
                    int userCount = 0;
                    // Check if there are any users to display
                    LDAPMessage* result = nullptr;
                    string filter = "(objectClass=inetOrgPerson)";
                    string searchBase = "ou=users," + basePath;
                                          rc    =    ldap_search_ext_sA(ldap,
const_cast<char*>(searchBase.c_str()),                  LDAP_SCOPE_ONELEVEL,
const_cast<char*>(filter.c_str()),  nullptr,  0,  nullptr,  nullptr,  nullptr,
LDAP_NO_LIMIT, &result);
                    if (rc == LDAP_SUCCESS)
                    {
                        userCount = ldap_count_entries(ldap, result);
                    }
                    ldap_msgfree(result);

                    if (userCount == 0)
                    {
                        cout << "There are no users to view. Try adding users
to the directory first." << endl;
                    }
                    else
                    {
                        while (true)
                        {
                            // View single/all existing users
                            string viewChoice;
                                cout  <<  "View  a  single  user  or  all  users?
(single/all): ";
                            getline(cin, viewChoice);

                            if (viewChoice == "single")
                            {
                                string userId;
                                cout << "Enter the user ID (cn): ";
                                getline(cin, userId);
                               string userDN = "cn=" + userId + ",ou=users,"
+ basePath;

                                displaySingleLDAPUser(ldap, userDN);
                                break;
                            }
                            else if (viewChoice == "all")
                            {
                                displayAllLDAPUsers(ldap, basePath);
                                break;
                            }
                            else
                            {
                               cout << "Invalid choice. Please enter 'single'
or 'all'." << endl;
                            }
                        }
                    }
                }
                else if (choice == "3")
                {
                    int userCount = 0;
                    // Check if there are any users to delete
                    LDAPMessage* result = nullptr;
                    string filter = "(objectClass=inetOrgPerson)";
                    string searchBase = "ou=users," + basePath;
                                           rc    =    ldap_search_ext_sA(ldap,
const_cast<char*>(searchBase.c_str()),                   LDAP_SCOPE_ONELEVEL,
const_cast<char*>(filter.c_str()),  nullptr,  0,  nullptr,  nullptr,  nullptr,
LDAP_NO_LIMIT, &result);
                    if (rc == LDAP_SUCCESS)
                    {
                        userCount = ldap_count_entries(ldap, result);
                    }
                    ldap_msgfree(result);

                    if (userCount == 0)
                    {
                          cout << "There are no users to delete. Try adding
users to the directory first." << endl;
                    }
                    else
                    {
                        while (true)
                        {
                            // Delete single/all existing users
                            string deleteChoice;
                                cout << "Delete a single user or all users?
(single/all): ";
                            getline(cin, deleteChoice);

                            if (deleteChoice == "single")
                            {
                                string userId;
```

```
                                 cout << "Enter the user ID (cn): ";
                                 getline(cin, userId);
                                string userDN = "cn=" + userId + ",ou=users,"
+ basePath;
                                 rc = deleteSingleLDAPUser(ldap, userDN);
                                 break;
                             }
                             else if (deleteChoice == "all")
                             {
                                 rc = deleteAllLDAPUsers(ldap, basePath);
                                 if (rc != LDAP_SUCCESS)
                                 {
                                     cerr << "Error deleting LDAP users under
base path '" << basePath << "'" << endl;
                                 }
                                 else
                                 {
                                       cout << "Deleted all LDAP users under
base path '" << basePath << "'" << endl;
                                 }
                                 break;
                             }
                             else
                             {
                                cout << "Invalid choice. Please enter 'single'
or 'all'." << endl;
                             }
                         }
                     }
                 }
                 else if (choice == "4")
                 {
                     // Exit
                     break;
                 }
                 else
                 {
                     cout << "Invalid choice. Please enter a valid option." <<
endl;
                 }
             }

             // Clean up
             cout << "Unbinding from LDAP server..." << endl;
             ldap_unbind_s(ldap);
             cout << "LDAP unbind successful. Connection closed." << endl;
         }
         else if (connectChoice == "n" || connectChoice == "no")
         {
             cout << "Are you sure you want to exit? (y/n): ";
             string exitChoice;
             getline(cin, exitChoice);
                           transform(exitChoice.begin(),    exitChoice.end(),
exitChoice.begin(), ::tolower);
             if (exitChoice == "y" || exitChoice == "yes")
             {
                 cout << "Exiting the program. Goodbye!" << endl;
```

```cpp
                break;
            }
            else if (exitChoice == "n" || exitChoice == "no")
            {
                continue;
            }
            else
            {
                cout << "Invalid choice. Please enter 'y' or 'n'." << endl;
            }
        }
        else
        {
            cout << "Invalid choice. Please enter 'y' or 'n'." << endl;
        }
    }

    return 0;
}
```

IX. Conclusion and Future Use

The LDAP User Management Application effectively addresses the critical need for efficient and secure user information management within organizations. By leveraging the LDAP protocol, the application ensures centralized storage and easy access to user data, which is essential for maintaining an organized and secure directory of user information.

Benefits to the Organization:

1. **Centralized User Management:** The application provides a single point of control for managing user information. This centralization simplifies administration, reduces redundancy, and ensures consistency across the organization.
2. **Scalability:** The use of LDAP allows the application to scale with the organization's growth. It can handle many users and extensive hierarchical data structures, making it suitable for organizations of any size.
3. **Efficiency in Bulk Operations:** By utilizing CSV files for bulk user operations, the application significantly reduces the time and effort required to add, view, and delete users. This feature is particularly beneficial during onboarding processes or mass updates.
4. **Enhanced Security:** Centralized user management through LDAP enhances security by ensuring that user data is stored and accessed in a controlled and secure manner. The application's authentication mechanisms help protect against unauthorized access.
5. **Error Handling and Validation:** The application includes robust error handling and validation mechanisms, ensuring that only valid user data is processed. This reduces the risk of errors and ensures data integrity.
6. **Cost Savings:** Automating user management processes reduces the administrative burden on IT staff, leading to cost savings in terms of time and resources. The efficiency gained from bulk operations further contributes to operational savings.

How Organizations Can Use This Application:

1. **Onboarding New Employees:** HR departments can use the application to quickly add new employees to the LDAP directory by importing their information from CSV files. This streamlines the onboarding process and ensures that new hires have access to necessary resources from day one.
2. **Maintaining User Information:** IT administrators can use the application to update and maintain user information, ensuring that the LDAP directory remains current and accurate. This is crucial for managing access rights and ensuring compliance with organizational policies.
3. **Access Control Management:** By maintaining an up-to-date LDAP directory, organizations can effectively manage access control for various applications and services. The application ensures that only authorized users have access to sensitive information and resources.
4. **Directory Cleanup:** The application can be used to periodically clean up the LDAP directory by deleting outdated or inactive user accounts. This helps maintain an organized directory and improves system performance.
5. **Integration with Other Systems:** Organizations can integrate the application with other identity management and directory services, creating a unified user management solution that spans multiple systems and platforms.

Future Enhancements:

1. **Enhanced User Interface:** Developing a graphical user interface (GUI) would provide a more intuitive and user-friendly experience, making it easier for non-technical users to manage LDAP records.

2. **Advanced Error Reporting:** Implementing detailed error reporting and logging mechanisms will help administrators diagnose and resolve issues more efficiently, enhancing overall system reliability.

3. **Automated Backup and Recovery:** Adding features for automated backup and recovery of user data will ensure data protection and quick restoration in case of system failures.

4. **Support for Additional Attributes:** Expanding the application to support additional LDAP attributes and customization options will allow organizations to tailor the user management process to their specific needs.

5. **Performance Optimization:** Optimizing the application's performance to handle larger datasets and more complex operations will improve efficiency and responsiveness.

By implementing these enhancements, the LDAP User Management Application can become an even more powerful tool for organizations, providing comprehensive solutions for user data management, access control, and directory maintenance.

X. References

- [1] Howes, T., & Smith, M. (1995). **The LDAP Application Programmer's Guide**. University of Michigan. Available: https://tools.ietf.org/html/rfc1823.
- [2] Wahl, M., Howes, T., & Kille, S. (1997). **Lightweight Directory Access Protocol (v3)**. Internet Engineering Task Force (IETF). Available: https://tools.ietf.org/html/rfc2251.
- [3] IETF (2006). **Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map**. Internet Engineering Task Force (IETF). Available: https://tools.ietf.org/html/rfc4510.
- [4] Microsoft. **Winldap.h header**. Available: https://learn.microsoft.com/en-us/windows/win32/api/winldap/.
- [5] Microsoft. **Windows API documentation**. Available: https://docs.microsoft.com/en-us/windows/win32/api/.
- [6] Microsoft. **CSV File Handling**. Available: https://docs.microsoft.com/en-us/cpp/standard-library/fstream.
- [7] Stroustrup, B. (2013). **The C++ Programming Language** (4th ed.). Addison-Wesley.
- [8] Josuttis, N. M. (2012). **The C++ Standard Library: A Tutorial and Reference** (2nd ed.). Addison-Wesley.
- [9] ISO/IEC. **ISO/IEC 14882:2011 - Information technology — Programming languages — C++**. Available: https://www.iso.org/standard/50372.html.
- [10] Koumoutsos, P., & Shekhar, R. (2004). **Managing LDAP directories**. IBM Redbooks. Available: https://www.redbooks.ibm.com/abstracts/sg246217.html.
- [11] OpenLDAP Foundation. **OpenLDAP Software 2.4 Administrator's Guide**. Available: https://www.openldap.org/doc/admin24/.
- [12] PracticeTestAutomation. **Practice Test Login**. Available: https://practicetestautomation.com/practice-test-login/.
- [13] OpenLDAP. **OpenLDAP Software Downloads**. Available: https://www.openldap.org/software/download/.
- [14] Canonical. **OpenLDAP - Ubuntu Package**. Available: https://packages.ubuntu.com/search?keywords=openldap&searchon=names&suite=all&section=all.
- [15] OpenLDAP Project. **Installing and Configuring OpenLDAP**. Available: https://www.openldap.org/doc/admin24/install.html.
- [16] Microsoft. **LDAP Client API Functions**. Available: https://learn.microsoft.com/en-us/windows/win32/api/_ldap/.
- [17] IBM. **Using LDAP to Manage User Authentication and Authorization**. Available: https://www.ibm.com/docs/en/was/9.0.5?topic=management-using-ldap.
- [18] Apache Directory. **Understanding LDAP**. Available: https://directory.apache.org/fortress/committer/ldap.html.
- [19] Red Hat. **LDAP Authentication**. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/identity_management_guide/configuring-ldap-authentication.